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

Планирование и анализ эксперимента

Лабораторная работа №2

Факультет: ФПМИ
Группа: ПМ-63
Студенты: Кожекин М.В.
Майер В.А.
Назарова Т.А.
Утюганов Д.С.
Вариант: 9(1)

Новосибирск
2020

1. Цель работы

Изучить алгоритмы, используемые при построении непрерывных оптимальных планов эксперимента.

2. Задание

1. Изучить условия оптимальности планов эксперимента и алгоритмы синтеза непрерывных оптимальных планов эксперимента.

2. Разработать программу построения непрерывных оптимальных планов эксперимента, реализующую последовательный или комбинированный алгоритм. Применить программу для построения оптимального плана для тестового примера из варианта заданий. Для отчета предусмотреть выдачу на печать протокола решения по итерациям. При большом числе итераций предусмотреть вывод протокола с некоторой дискретностью.

3. Оформить отчет, включающий в себя постановку задачи, протокол решения, графическое изображение начального плана и полученного оптимального плана, а также текст программы.

4. Защитить лабораторную работу.

3. Анализ

Задана двухфакторная модель на квадрате [-1, 1].

$$y = \Theta_0 + \Theta_1 \cdot x_1 + \Theta_2 \cdot x_2 + \Theta_3 \cdot x_1 \cdot x_2 + \Theta_4 \cdot x_1^2 + \Theta_5 \cdot x_2^2$$

Начальный план - полный двухфакторный эксперимент из 25 точек, на уровнях -1, -0.5, 0, +0.5, +1, веса равны 1/25. Строить D-оптимальные планы. Последовательный алгоритм.

Этапы **последовательного алгоритма** синтеза непрерывного оптимального плана:

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

2. Отыскивается точка глобального эстремума x^s:

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

3. Проверяется приближенное выполнение необходимых и достаточных условий оптимальности планов

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

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

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

В конце происходит процедура "очистки" плана.

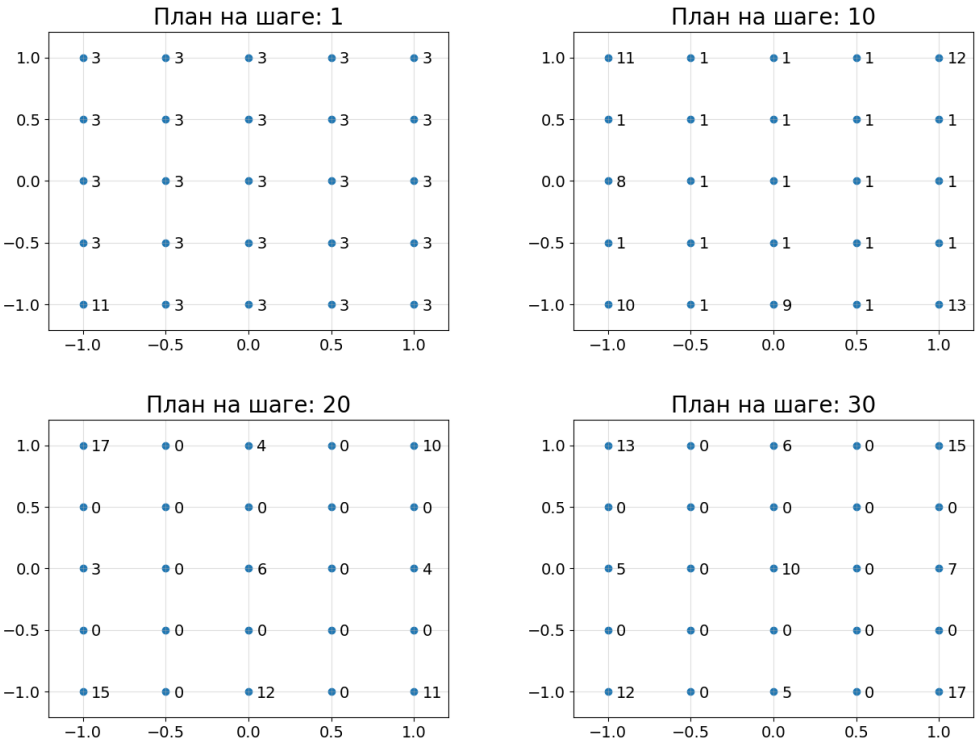
1. Точки, тяготеющие к одной из групп, объединяются по правилу

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

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

4. Визуализация работы алгоритма

На каждой точке спектра плана указан её вес в процентах: Как видно решение сходится к плану, с координатами точек [-1, 0, 1].



5. Исходный код программы

lab2.py

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


plt.rcParams.update({'font.size': 14})
k = 2       # число переменных 2: (x,y)
m = 6       # число параметров a + b*x + c*y + d*x*y + e*x^2 + f*y^2
n = 25      # число точек сетки
width = 5   # сторона сетки

def f(theta, x):
    return  theta[0] + \
            theta[1]*x[0] + \
            theta[2]*x[1] + \
            theta[3]*x[0]*x[1] + \
            theta[4]*x[0]**2 + \
            theta[5]*x[1]**2

def f_vector(x):
    return np.array([
        [1],
        [x[0]],
        [x[1]],
        [x[0]*x[1]],
        [x[0]**2],
        [x[1]**2]
    ])

def f_vector_T(x):
    return np.array([
        1,
        x[0],
        x[1],
        x[0]*x[1],
        x[0]**2,
        x[1]**2
    ])


#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
class Lab2():
    '''
    Класс для 2 лабораторной работы
    Для стандартизации все критерии ищут минимальное значение
    '''
    def __init__(self):
        ''' Выделение памяти под массивы '''
        self.x = np.ndarray((n, k))
        self.p = np.ndarray(n)
        self.new_x = np.ndarray((n, k))
        self.new_p = np.ndarray(n)
        self.t = np.linspace(-1, 1, width)
        self.M = np.ndarray((m, m))
```

```
        self.alpha = 1 / n
        self.gamma = 2
        self.max_iter_s = 30
        self.max_iter_alpha = 20

    def generate_initial_guess(self):
        ''' Задаём начальное приближение '''
        i = 0
        for x1 in self.t:
            for x2 in self.t:
                self.x[i] = np.array([x1, x2])
                i+=1
        self.p = np.full(n, 1/n)

    def fi(self, x):
        ''' Значение функции fi в точке x '''
        return f_vector_T(x) @ self.D @ f_vector(x)

    def max_fi(self):
        ''' Поиск максимального значения fi '''
        max_fi = -9000
        for point in self.x:
            fi = self.fi(point)
            if fi > max_fi:
                max_fi = fi
        return max_fi

    def is_plan_optimal(self):
        '''
        Проверяем выполнение необходимых и достаточных
        условий оптимальности плана
        '''
        max_fi = self.max_fi()
        delta = 0.01 * abs(max_fi)
        if abs(-max_fi + np.trace(self.M @ self.D)) <= delta:
            return True
        else:
            return False

    def clear_plan(self):
        ''' Процедура очистки плана '''
        global n
        i = 0
        for x1 in self.t:
            for x2 in self.t:
                self.new_x[i] = np.array([x1, x2])
                i+=1
        self.new_p = np.zeros(width**2)

        for point, weigth in zip(self.x, self.p):
            for i in range(width):
                for j in range(width):
                    a = point
                    b = self.new_x[i*width+j]
                    if a[0]==b[0] and a[1]==b[1]:
                        self.new_p[i*width+j] += weigth
        self.x = np.copy(self.new_x)
        self.p = np.copy(self.new_p)
        n = width**2

```

```
    def calc_new_point(self):
        ''' Выбираем новую точку плана '''
        max_fi = -9000
        new_point = self.x[0]
        for point in self.x:
            fi = self.fi(point)
            if fi > max_fi:
                max_fi = fi
                new_point = point
        return new_point

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

    def draw_plan(self, iteration):
        ''' Отрисовка весов плана эксперимента '''
        x, y = np.hsplit(self.x, 2)
        plt.scatter(x, y)
        for i, txt in enumerate(self.p):
            plt.annotate(str(int(txt*100)), (x[i] + 0.05, y[i] - 0.05))
        plt.xticks(self.t)
        plt.yticks(self.t)
        plt.title('План на шаге: ' + str(iteration), fontsize=20)
        plt.grid(alpha=0.4)
        plt.margins(0.1)
        plt.savefig('report/plan' + str(iteration) + '.png')
        plt.clf()

    def sequential_algorithm(self):
        '''
        Последовательный алгоритм синтеза непрерывного
        оптимального плана эксперимента
        '''
        self.generate_initial_guess()
        do_calc = True
        s = 0

        while do_calc == True and s < self.max_iter_s:
            a = 0
            self.alpha = 1 / n
            self.build_matrix_M()
            self.build_matrix_D()
            psi = self.calc_D()
            self.add_new_point()
            psi_next = self.calc_D()

            # Уменьшаем шаг, если метод расходится
            while psi_next >= psi and a < self.max_iter_alpha:
                a += 1
                self.alpha /= self.gamma
                psi = psi_next
                self.add_new_point()
                psi_next = self.calc_D()

            print(s+1, a)
```

```
            self.clear_plan()
            self.draw_plan(s+1)
            do_calc = not self.is_plan_optimal()
            s += 1

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

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

    def calc_D(self):
        '''
        Критерий D – оптимальности. (D – determinant)
        Эллипсоид рассеивания имеет минимальный объём
        '''
        return np.log(det(self.M))



#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------

l2 = Lab2()
l2.sequential_algorithm()
```