# Министерство образования и науки Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»

NSTU_Logo_blue

## Кафедра теоретической и прикладной информатики

### Лабораторная работа № 2 по дисциплине «Математические методы оптимального планирования эксперимента»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| сигма градиент синий1 | Факультет: | ПМИ |  |  |
| Группа: | ПМИМ-01 |  |  |
| Студенты: | Ершов П.К., Шибалова Ю.В. |  |  |
| Вариант: | 5 |  |  |
| Преподаватель: | Попов А.А. |  |  |

Новосибирск

2020

1. **Цель работы**

Изучить алгоритмы, используемые при построении непрерывных оптимальных планов эксперимента.

1. **Задание**
2. Изучить условия оптимальности планов эксперимента и алгоритмы синтеза непрерывных оптимальных планов эксперимента.
3. 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.
4. Оформить отчет, включающий в себя постановку задачи, протокол решения, графическое изображение начального плана и полученного оптимального плана, а также текст программы.
5. Защитить лабораторную работу.

Двухфакторная квадратичная модель на квадрате [–1, +1]. Начальный план – полный факторный эксперимент из 25 точек, на уровнях –1, –0.5, 0, +0.5, +1, веса равны 1/25. Строить Φ2 -оптимальные планы. Последовательный алгоритм.

|  |  |  |
| --- | --- | --- |
| Критерий | Функционал | Необходимые и достаточные условия оптимальности |
|  |  |  |

1. **Описание алгоритма работы**

Построение непрерывного Ψ-оптимального плана состоит в решении экстремальной задачи вида

Последовательный алгоритм:

Основная часть алгоритма:

1. Выбираем невырожденный план и номер итерации
2. Отыскиваем глобальный экстремум : , где =
3. Проверяем необходимое и достаточное условие оптимальности, если выполняется, то заканчиваем:

*, где*

1. Составляем план . Пересчитываем веса и добавляем с весом .
2. Делаем сравнение, если , то уменьшаем в два раза и переходим на шаг 4,иначе s заменяем на s+1 и осуществляем переход на шаг 2

Очистка плана:

Проверяем:

1) , , где μ – малое положительное число

2), где π – малое положительное число;

3) план по сравнению с планом имеет "посторонние" точки с малыми весами ;

4) вместо одной точки, близкой к, имеется набор точек , каждая из которых близка к : и их суммарный вес близок к

Производим очистку:

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

2. Точки с малыми весами, не тяготеющие ни к одной из групп, указанных в п.4 выбрасываются. Их веса перераспределяются между оставшимися точками. (удаление)

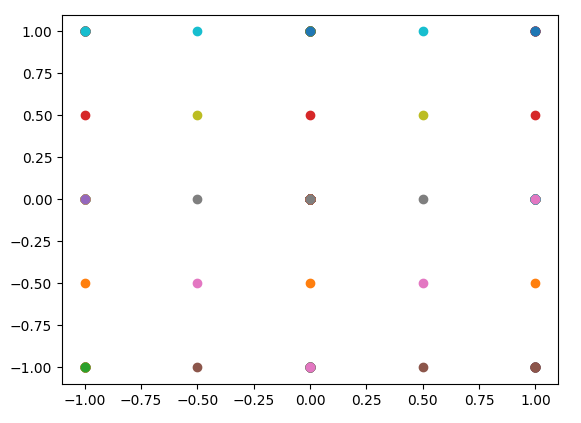
После этого проверяем очищенный план на оптимальность.

1. **Ход работы**

План после первой итерации.

|  |  |
| --- | --- |
| Точки | Веса точек |
| [-1, -1]  [-1, -0.5]  [-1, 0]  [-1, 0.5]  [-1, 1]  [-0.5, -1]  [-0.5, -0.5]  [-0.5, 0]  [-0.5, 0.5]  [-0.5, 1]  [0, -1]  [0, -0.5]  [0, 0]  [0, 0.5] | 0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054  0.00992054 |
| …… | ……. |
| [-1.0, 1]  [-1.0, 0]  [0, 0]  [0, 1]  [-1.0, -1.0]  [1, -1.0]  [1, 0]  [0, 0]  [0, -1.0]  [0, 0]  [-1.0, 0]  [-1.0, 1]  [0, 0] | 0.00096532  0.00096437  0.00096344  0.00096250  0.00096157  0.00096063  0.00095971  0.00095878  0.00095786  0.00095694  0.00095602  0.00095511  0.00095419 |

Суммарное количество точек плана до очистки: 525

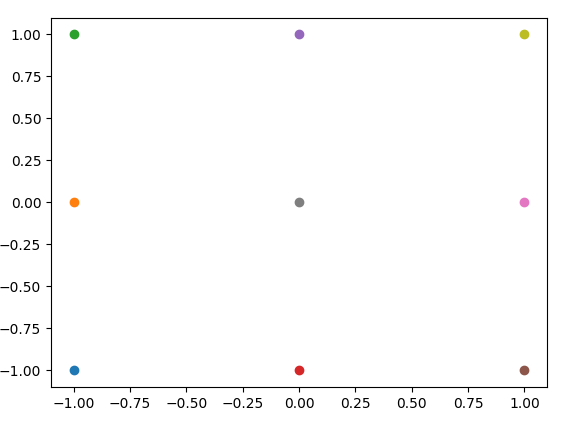
****

**Рисунок 1. График точек до очистки плана на первой итерации**

План после очистки:

|  |  |
| --- | --- |
| Точки | Веса точек |
| [-1, -1]  [-1, 0]  [-1, 1]  [0, -1]  [0, 1]  [1, -1]  [1, 0]  [0, 0]  [1, 1] | 0.07830861669954924  0.10662257179420202  0.0789809200562427  0.09250783974362378  0.10582630202184257  0.08092050276526429  0.1011686739225718  0.27571226317735753  0.07995230981934637 |

Количество точек после очистки плана: 9.

****

**Рисунок 2. График точек после очистки плана**

1. **Код программы**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
N = 6 *# размерность информационной матрицы*p = 2 *# параметр для Ф2-оптимальности*xy = [-1, -0.5, 0, 0.5, 1] *# исходный массив для формирования точек плана*grig = np.arange(-1, 1.05, 0.01) *# сетка с малым шагом для поиска экстремума*X = [] *# массив точек плана*X\_G = [] *# массив точек для поиска экстремума*gamma = 2 *# параметр уменьшения коэффициента альфа***for** x **in** xy: *# получаем начальный план* **for** y **in** xy:  
 X.extend([[x,y]])  
  
**for** x **in** grig: *# получаем план для поиска экстремума* **for** y **in** grig:  
 X\_G.extend([[x,y]])  
  
plan\_w = [1 / 25] \* 25 *# веса точек в плане***def** f\_model(x, y): *# функция модели* **return** [1, x, y, x\*y, x\*x, y\*y]  
  
**def** M\_mat(x\_p, p): *#Информационную матрицу M* M = np.zeros(shape = (N, N))  
 **for** i **in** range(N):  
 **for** j **in** range(N):  
 **for** k **in** range(len(x\_p)):  
 f = np.array(f\_model(x\_p[k][0], x\_p[k][1]))  
 M[i][j] += p[k] \* f[i] \* f[j]  
 **return** M  
  
  
**def** find\_ext(M, X): *# функция поиска экстремума* fi = [] *# массив значений экстремума* M1 = np.linalg.inv(M) *# получаем обратную матрицу от информационной матрицы M* M3 = np.linalg.matrix\_power(M1, 3) *# получаем обратную матрицу в кубе (так как в формуле для поиска экстремума есть  
 # дифференциал от функцилнала критерия Ф2 по информационной матрице  
 # то, при вычислении дифференциала получаем обратную матрицу в степени 3)* **for** i **in** X: *# получаем набор чисел для поиска экстремума (в данном варианте - наибольнее число)* f\_i = np.array(f\_model(i[0], i[1])) *# получаем вектор-столбец от функции модели* buf = f\_i.T.dot(M3) *# умножаем вектор-строку от функции модели на обратную матрицу в кубе* fi.append(buf.dot(f\_i))*# умножаем получившийся в предыдущем действии вектор на вектор-столбец от функции модели* max\_e = max(fi)  
 **for** i **in** range(len(X)):  
 **if**(max\_e == fi[i]):  
 **return** max\_e, X[i] *# возвращаем наибольшее значение (экстремум) и точку экстремума***def** nsc\_m(M, coef): *# проверяем выполнение приближённых необходимых и достаточных условий оптимальности* M1 = np.linalg.inv(M) *# получаем обратную матрицу от информационной матрицы M* M3 = np.linalg.matrix\_power(M1, 3) *# получаем обратную матрицу в кубе (дифференциал от функционала по информационной матрице)* trM = np.trace(M.dot(M3)) *# получаем след от произведения информационной матрицы на обратную матрицу в кубе* ext, buf = find\_ext(M, X) *# получаем экстремум* sig = abs(ext) \* coef *# получаем параметр сигма* **if** ( abs(trM - ext) <= sig): *# если модуль разности следа матрицы и экстремума меньше или равен сигме, то НиД выполнены и план оптимален* **return** 1  
 **else**:  
 **return** 0  
  
**def** fix\_error(x): *# функцяи удаления погрешности при копировании точек* **for** i **in** range(len(x)):  
 **if** x[i] < -1:  
 x[i] = -1  
 **if** x[i] < -0.75 **and** x[i] > -1:  
 x[i] = -1  
 **if** x[i] > 1:  
 x[i] = 1  
 **if** x[i] < 1 **and** x[i] > 0.75:  
 x[i] = 1  
 **if** x[i] < 0 **and** x[i] > -0.25:  
 x[i] = 0  
 **if** x[i] > 0 **and** x[i] < 0.25:  
 x[i] = 0  
 **if** x[i] < -0.5 **and** x[i] > -0.75:  
 x[i] = -0.5  
 **if** x[i] > -0.5 **and** x[i] < -0.25:  
 x[i] = -0.5  
 **if** x[i] > 0.5 **and** x[i] < 0.75:  
 x[i] = 0.5  
 **if** x[i] < 0.5 **and** x[i] > 0.25:  
 x[i] = 0.5  
 **return** x  
  
**def** cr\_new\_plan(M, X, plan\_w, x\_ext): *# функция получения нового плана и новых весов плана* new\_plan\_w = [] *# массив новых весов плана* a = 1 / len(plan\_w) *# коэффициент альфа* X.append(x\_ext) *# добавляем новую точку (точку экстремума) в план* f = 0  
 **while** (f != 1): *# пока функционал от ного плана меньше или равен функционалу от старого плана пересчитываем веса плана* new\_plan\_w = plan\_w.copy()  
 **for** i **in** range(len(plan\_w)): *# получаем новые веса плана* new\_plan\_w[i] = new\_plan\_w[i] \* (1 - a)  
 new\_plan\_w.append(a) *# увеличиваем массив весов плана* f = test\_M(M, X, new\_plan\_w) *# проверяем условие выхода из цикла* M = M\_mat(X, new\_plan\_w) *# получаем новую информационную матрицу по новому плану и новым весам точек* a /= gamma *# если условие не выполняется - уменьшаем альфу в 2 раза и вычисляем новые веса плана* **return** X, new\_plan\_w  
  
**def** test\_M(M, xs\_1, p\_1): *# сравниваем функционалы от старой и новой информационных матриц* M\_n = M\_mat(xs\_1, p\_1)  
 M1\_n = np.linalg.inv(M\_n) *# получаем обратную матрицу от новой информационной матрицы M\_n* M2\_n = np.linalg.matrix\_power(M1\_n, 2) *# получаем обратную матрицу в квадрате* M1 = np.linalg.inv(M) *# получаем обратную матрицу от информационной матрицы M* M2 = np.linalg.matrix\_power(M1, 2) *# получаем обратную матрицу в квадрате* FM\_old = - (p \*\* (-1)) \* np.trace(M2)  
 FM\_new = - (p \*\* (-1)) \* np.trace(M2\_n)  
 **if**(FM\_new < FM\_old):  
 **return** 1  
 **else**:  
 **return** 0  
  
**def** iter\_alg(M, X, plan\_w): *# итерационный алгоритм* **while True**: *# увеличиваем план до тех пор, пока новый план не будет близким к оптимальному* ext, x\_ext = find\_ext(M, X\_G) *# получаем экстремуму и точку экстремума* x\_ext = fix\_error(x\_ext) *# убираем погрешности от копирования* **if** (nsc\_m(M, 0.08)): *# проверяем оптимальность плана и если он оптимален - возвращаем новый план и веса точек* **return** X, plan\_w  
 X, plan\_w = cr\_new\_plan(M, X, plan\_w, x\_ext) *# добавляем новую точку в план и пересчитываем веса точек* M = M\_mat(X, plan\_w) *# получаем новую иформационную матрицу* **return** X, plan\_w  
  
**def** findClose(x, X): *# поиск точек, близких к другим точкам* **for** i **in** range(len(X)):  
 vec = np.array([x[0] - X[i][0], x[1] - X[i][1]])  
 scal = np.dot(vec, vec)  
 **if** np.sqrt(scal)\*\*2 < 0.05:  
 **return** i  
 **return** -1  
  
**def** unionCloseDots(x, p):*#объединение близких точек* newX = [x[0]]  
 newP = [p[0]]  
 **for** i **in** range(1, len(x)):  
 index = findClose(x[i], newX)  
 **if** index == -1:  
 newX.append(x[i])  
 newP.append(p[i])  
 **else**:  
 newP[index] += p[i]  
 x = newX  
 p = newP  
 **return** x, p  
  
**def** removeDots(x, p):*#удаление точек с малыми весами* sum = 0  
 index = 0  
 **for** i **in** range(len(p)):  
 **if** p[i] < 0.03:  
 sum += p[i]  
 p[i] = 0  
 x[i] = [0, 0]  
 index += 1  
 **for** i **in** range(index):  
 p.remove(0)  
 x.remove([0, 0])  
 sum /= len(p)  
 **for** i **in** range(len(p)):  
 p[i] += sum  
 **return** x, p  
  
**def** main(X, plan\_w): *# главная функция* s = 0  
 M = M\_mat(X, plan\_w) *# получаем начальную информационную матрицу* **while** (nsc\_m(M, 0.1) == 0): *# проверка плана на оптимальность* M = M\_mat(X, plan\_w)  
 X\_n, P\_n = iter\_alg(M, X, plan\_w) *# получаем новый план* **if**(s == 0):  
 **for** i **in** X\_n:  
 plt.scatter(i[0], i[1])  
 plt.show()  
 print(**"X P"**)  
 **for** i **in** range(len(X\_n)):  
 print(X\_n[i], P\_n[i])  
 print(**"Количество точек до очистки плана = "**, len(X\_n))  
 X\_m, p\_m = unionCloseDots(X\_n, P\_n) *# объединяем близкие точки* X, plan\_w = removeDots(X\_m, p\_m) *# очищаем план от лишних точек* print(**"Количество точек после очистки плана = "**, len(X))  
 **if** (s == 0):  
 **for** i **in** X:  
 plt.scatter(i[0], i[1])  
 plt.show()  
 print(**"X P"**)  
 **for** i **in** range(len(X)):  
 print(X[i], plan\_w[i])  
 s += 1  
 **return** X, plan\_w  
  
  
X, P = main(X, plan\_w)  
**for** i **in** X:  
 plt.scatter(i[0], i[1])  
plt.show()  
print(**"X P"**)  
**for** i **in** range(len(X)):  
 print(X[i], P[i])