|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
|  | | |
| Лабораторная работа № 2 | | |
| по дисциплине «Математические методы оптимального планирования эксперимента» | | |
|  | | |
| **построение непрерывных оптимальных планов эксперимента** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИМ-21 |
| Вариант: | 3 |
| Студенты: | Демидович Е. |
|  | Стародубцев С. |
|  | Цыганков А. |
| Преподаватель: | Попов А.А. |
|  |  |
|  | | |
| Новосибирск | | |
| 2022 | | |

# 1. Цель работы

Изучить алгоритмы, используемые при построении непрерывных оптимальных планов эксперимента.

# 2. Содержание работы

1. Изучить условия оптимальности планов эксперимента и алгоритмы синтеза непрерывных оптимальных планов эксперимента.
2. Разработать программу построения непрерывных оптимальных планов эксперимента, реализующую последовательный алгоритм. Применить программу для построения оптимального плана для тестового примера из варианта заданий. Для отчёта предусмотреть выдачу на печать протокола решения по итерациям. При большом числе итераций предусмотреть вывод протокола с некоторой дискретностью.
3. Оформить отчёт, включающий в себя постановку задачи, протокол решения, графическое изображение начального плана и полученного оптимального плана, а также текст программы.
4. Защитить лабораторную работу.

# 3. Постановка задачи

Исследуемая модель имеет вид:

где – значение зависимой переменной,

– заданная вектор-функция от независимой векторной переменной , которая может изменяться на заданном отрезке [-1, 1],

– вектор неизвестных параметров, которые необходимо определить по результатам экспериментов (измерений),

– ошибка, распределённая по нормальному закону.

Критерий А-оптимальности:



Необходимое и достаточное условие A-оптимальности:

Заданная вектор-функция:

Исходный план – полный факторный эксперимент из 49 точек, на уровнях –1, –0.75, –0.25, 0, +0.25, +0.75, +1, веса равны 1/49:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | -1 | -1 | … | 1 | 1 |
|  | -1 | -0.75 | … | 0.75 | 1 |
|  | 0.02041 | 0.02041 | … | 0.02041 | 0.02041 |

Для построения оптимального плана будем использовать последовательный алгоритм.

Шаг 1 - 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

Шаги 2 и 3 - очистка плана и удаление точек с малыми весами:

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

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

После этого проверяем очищенный план на оптимальность.

# 4. Ход выполнения

Шаг 1 – основная часть алгоритма

Протокол выполнения

|  |  |  |  |
| --- | --- | --- | --- |
| ***Номер итерации*** | ***Длина плана*** |  | ***d*** |
| 1 | 50 | 1.03263723 | 76.67195912 |
| 250 | 534 | 0.1983171 | 1.03761762 |
| 500 | 1034 | 0.19054281 | 0.51956565 |
| 750 | 1534 | 0.18787288 | 0.35287603 |
| 1000 | 2034 | 0.18688175 | 0.3083739 |
| 1250 | 2534 | 0.18601181 | 0.25631119 |
| 1500 | 3034 | 0.18526003 | 0.20564919 |
| 1540 | 3134 | 0.18498881 | 0.18179114 |

Проверка на A-оптимальность сгенерированного плана:



Шаг 2 – очистка плана

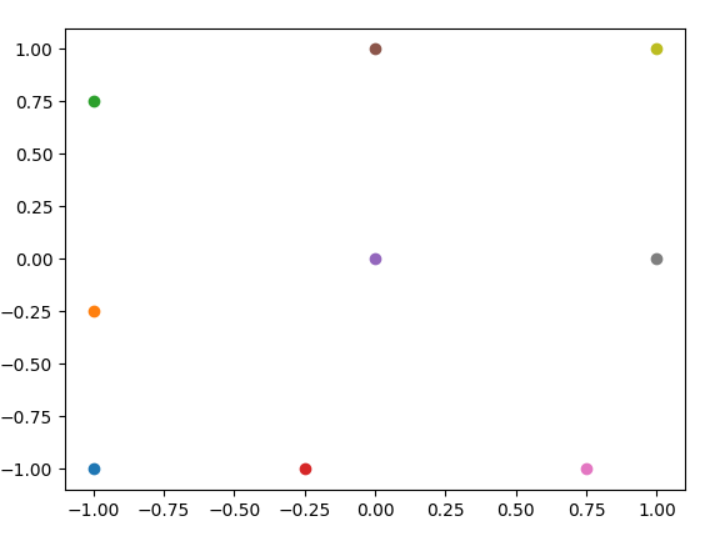
|  |  |  |
| --- | --- | --- |
| ***x1*** | ***x2*** | ***p*** |
| -1 | -1 | 0.09308292 |
| -1 | -0.25 | 0.09554764 |
| -1 | 0.25 | 0.00166054 |
| -1 | 0.75 | 0.09350551 |
| -0.75 | -0.75 | 0.00083027 |
| -0.75 | 0 | 0.00083027 |
| -0.75 | 1 | 0.00083027 |
| -0.25 | -1 | 0.09518052 |
| -0.25 | -0.25 | 0.00249081 |
| -0.25 | 0.25 | 0.00166054 |
| -0.25 | 0.75 | 0.00249081 |
| 0 | -0.75 | 0.00166054 |
| 0 | 0 | 0.22239221 |
| 0 | 1 | 0.09573963 |
| 0.25 | -1 | 0.00083027 |
| 0.25 | -0.25 | 0.00083027 |
| 0.25 | 0.25 | 0.00083027 |
| 0.25 | 0.75 | 0.00083027 |
| 0.75 | -1. | 0.09385967 |
| 0.75 | -0.25 | 0.00249081 |
| 0.75 | 0.25 | 0.00166054 |
| 0.75 | 0.75 | 0.00249081 |
| 1 | -0.75 | 0.00083027 |
| 1 | 0 | 0.0948255 |
| 1 | 1 | 0.09261889 |

Шаг 3 – удаление точек с малыми весами

|  |  |  |
| --- | --- | --- |
| ***x1*** | ***x2*** | ***p*** |
| -1 | -1 | 0.09308291584028708 |
| -1 | -0.25 | 0.09554763643963925 |
| -1 | 0.75 | 0.09350550701473029 |
| -0.25 | -1 | 0.0951805191421396 |
| 0 | 0 | 0.2223922064137818 |
| 0 | 1 | 0.09573963154531294 |
| 0.75 | -1 | 0.09385966956921558 |
| 1 | 0 | 0.09482550224490265 |
| 1 | 1 | 0.09261889389655475 |
| -1 | -1 | 0.09308291584028708 |

Проверка очищенного плана на A-оптимальность:

График очищенного плана:



# Код программы

**import** json  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
m = 6  
gamma = 2  
  
  
*# f(x)***def** f(a\_, b\_):  
 f\_ = np.array([[1], [a\_], [b\_], [a\_ \* b\_], [a\_ \*\* 2], [b\_ \*\* 2]])  
 **return** f\_  
  
  
*# чтение плана из файла***def** read\_plan():  
 **with** open(**"data2.json"**, **"r"**) **as** file:  
 x1\_ = []  
 x2\_ = []  
 p\_ = []  
 plans = json.load(file)[**"plan"**]  
 **for** plan **in** plans:  
 x1\_.append(plan[**"x1"**])  
 x2\_.append(plan[**"x2"**])  
 p\_.append(plan[**"p"**])  
 **return** x1\_, x2\_, p\_  
  
  
*# Построение матрицы M***def** get\_mat\_m(c, d, p1, mat\_m):  
 mat\_m = np.zeros((mat\_m, mat\_m))  
 n = len(x1)  
 **for** q **in** range(0, n - 1):  
 mat\_m += p1[q] \* f(c[q], d[q]) @ np.transpose(f(c[q], d[q]))  
 **return** mat\_m  
  
  
*# Построение дисперсионной матрицы D***def** get\_mat\_d(mat\_m):  
 mat\_d = np.linalg.inv(mat\_m)  
 **return** mat\_d  
  
  
*# 2. найти глобальный экстремум***def** get\_global\_max(mat\_d):  
 phi\_max = -100  
 argmax\_x1 = 0  
 argmax\_x2 = 0  
 l1 = -1  
 i1 = -1  
 **while** i1 <= 1.0:  
 i2 = -1  
 **while** i2 <= 1.0:  
 phi = (np.transpose(f(i1, i2)) @ mat\_d @ mat\_d @ f(i1, i2)).item() *# A-plan* **if** phi\_max < phi:  
 phi\_max = phi  
 argmax\_x1 = i1  
 argmax\_x2 = i2  
 l1 += 1  
 i2 += 0.01  
 i1 += 0.01  
 **return** phi\_max, argmax\_x1, argmax\_x2  
  
  
*# 3. проверка необходимых и достаточных условий***def** check\_condition(phi\_, mat\_m, mat\_d):  
 delta = np.abs(phi\_) \* 0.01  
 print(**"delta = "**, delta)  
 d = np.abs(np.trace(mat\_d @ mat\_d @ mat\_m) - phi\_) *# A-plan* print(**"d = "**, d)  
 **return** d <= delta  
  
  
*# 4. составление нового плана***def** create\_plan(i1, i2, ps1, b1, b2, a1):  
 ns = len(ps1)  
 **for** w **in** range(0, ns):  
 ps1[w] = (1 - a1) \* ps1[w]  
  
 b1.append(i1)  
 b2.append(i2)  
 ps1 = np.append(ps1, a1)  
  
 print(**"len(x1) = "**, len(b1))  
 print(**"len(x2) = "**, len(b2))  
 print(**"len(p) = "**, len(ps1))  
 **return** ps1, b1, b2  
  
  
*# Создание графика***def** show\_scatter(a\_, b\_):  
 **for** i1 **in** range(0, len(a\_)):  
 plt.scatter(a\_[i1], b\_[i1])  
 plt.plot()  
 plt.show()  
  
  
x1, x2, p = read\_plan()  
  
*# 1. выбор плана e0, s = 0*s = 0  
print(**"План e"**, s)  
M = []  
D = []  
  
**while True**:  
 *# 2 - глобальный экстремум* M = get\_mat\_m(x1, x2, p, m)  
 D = get\_mat\_d(M)  
  
 phiMax, x1Max, x2Max = get\_global\_max(D)  
  
 *# 3 - проверка условий* **if** check\_condition(phiMax, M, D):  
 print(**"алгоритм оптимален"**)  
 print(**"x1 = "**, x1)  
 print(**"x2 = "**, x2)  
 print(**"p = "**, p)  
 **break** *# 4 - новый план* a = 1 / len(p)  
 p, x1, x2 = create\_plan(x1Max, x2Max, p, x1, x2, a)  
  
 *# 5 - сравнение функционала* Ms = get\_mat\_m(x1, x2, p, m)  
 Ds = get\_mat\_d(Ms)  
 **while True**:  
 **if** np.trace(Ds) > np.trace(D):  
 print(**"det(Ms) = "**, np.linalg.det(Ms), **"det(M) = "**, np.linalg.det(M))  
 a /= gamma  
 print(**"a = "**, a)  
 p, x1, x2 = create\_plan(x1Max, x2Max, p, x1, x2, a)  
 Ms = get\_mat\_m(x1, x2, p, m)  
 Ds = get\_mat\_d(Ms)  
 **else**:  
 **break** M = get\_mat\_m(x1, x2, p, m)  
 D = get\_mat\_d(M)  
 print(**"s = "**, s)  
 **if** s % 250 == 0:  
 print(**"len(x1) = "**, len(x1), **"x1:\n"**, x1)  
 print(**"len(x2) = "**, len(x2), **"x2:\n"**, x2)  
 print(**"len(p) = "**, len(p), **"p:\n"**, p)  
 print(**"Проверка на оптимальность - неочищенный план A"**)  
 print(**"Правая часть: "**, np.trace(D))  
 print(**"Левая часть: "**, np.trace(D @ D @ M))  
  
 s += 1  
  
M = get\_mat\_m(x1, x2, p, m)  
D = get\_mat\_d(M)  
  
*# Проверка на оптимальность*print(**"Первая проверка на оптимальность - неочищенный план A"**)  
print(**"Правая часть: "**, np.trace(D))  
print(**"Левая часть: "**, np.trace(D @ D @ M))  
  
newX1 = [0]  
newX2 = [0]  
newP = [0]  
newX1[0] = x1[0]  
newX2[0] = x2[0]  
newP[0] = p[0]  
mu = 0.1  
pi = 0.02  
k = len(x1)  
r = 0  
i = 1  
print(**"sum p = "**, np.sum(p))  
  
**while True**:  
 **if** r >= len(p) - 1:  
 **break  
  
 while** i < len(p):  
 scalar = (x1[r] - x1[i]) \*\* 2 + (x2[r] - x2[i]) \*\* 2  
 **if** scalar <= mu: *# проверка на близость* p[r] += p[i] *# добавление веса к точке* p = np.delete(p, i)  
 x1 = np.delete(x1, i)  
 x2 = np.delete(x2, i)  
 i -= 1  
 i += 1  
 r += 1  
 i = r + 1  
  
print(**"Выполняется очистка плана"**)  
  
print(**"len(x1) = "**, len(x1), **"x1:\n"**, x1)  
print(**"len(x2) = "**, len(x2), **"x2:\n"**, x2)  
print(**"len(p) = "**, len(p), **"p:\n"**, p)  
  
sumP = 0  
newX1 = x1  
newX2 = x2  
newP = p  
**for** i **in** range(1, len(newP)): *# найти в массиве р малые веса* **if** newP[i] <= pi:  
 sumP += newP[i] *# сумма малых весов* newP[i] = 0  
  
newX11 = []  
newX21 = []  
newP1 = []  
**for** i **in** range(0, len(newP)): *# удаление из массивов элементов с нулевыми весами* **if** newP[i] != 0:  
 newP1.append(newP[i])  
 newX11.append(newX1[i])  
 newX21.append(newX2[i])  
  
p = newP1  
x1 = newX11  
x2 = newX21  
  
sumP = sumP / len(p)  
  
print(**"План очищен!"**)  
M = get\_mat\_m(x1, x2, p, m)  
D = get\_mat\_d(M)  
  
*# Проверка на оптимальность*print(**"Вторая проверка на оптимальность - очищенный план A"**)  
print(**"Правая часть: "**, np.trace(D))  
print(**"Левая часть: "**, np.trace(D @ D @ M))  
show\_scatter(x1, x2)  
  
print(**"len(x1) = "**, len(x1), **"x1:\n"**, x1)  
print(**"len(x2) = "**, len(x2), **"x2:\n"**, x2)  
print(**"len(p) = "**, len(p), **"p:\n"**, p)