# 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 № 4 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 методы оптимального планирования эксперимента при нелинейной параметризации функции отклика.

1. **Задание**
2. Изучить понятия локально-оптимального планирования и информационной матрицы при нелинейной параметризации функции отклика, ознакомиться с видом производственной функции КоббаДугласа.
3. По заданному типу технологии сформировать имитационную модель в виде производственной функции Кобба-Дугласа. При этом задать истинные значения для параметров, нелинейно входящих в модель. Выход модели зашумить, уровень шума установить в пределах 15…20 % от мощности полезного сигнала.
4. Выбрать план для затравочного эксперимента, состоящий из небольшого числа наблюдений, и смоделировать на его основе экспериментальные данные.
5. Оценить параметры модели по полученным экспериментальным данным. Для этого необходимо перейти к линейной модели, воспользовавшись логарифмическим представлением уравнения модели наблюдения. Параметры преобразованной модели тогда можно оценить обычным «линейным» МНК.
6. Построить локально-оптимальный план эксперимента для исходной нелинейной модели, воспользовавшись разработанной ранее программой синтеза дискретных оптимальных планов и полученными оценками параметров модели. Число наблюдений должно в 4…5 раз превышать число параметров модели.
7. По сформированной ранее (п. 2) имитационной модели провести имитационный эксперимент в точках полученного локальнооптимального плана. Провести оценку параметров и вычислить норму отклонения оценок от их истинных значений. Вычислительный эксперимент повторить не менее 100 раз, каждый раз с новой реализацией помехи. Вычислить среднее значение нормы отклонения оценок. Процедуру повторить, используя в качестве плана эксперимента случайно расположенные точки в факторном пространстве. В серии вычислительных экспериментов случайный план фиксируется (выбирается один раз). Сделайте вывод об эффективности оптимального планирования эксперимента для идентификации заданной нелинейной модели.
8. Оформить отчет, включающий в себя постановку задачи, оценки параметров по затравочному эксперименту, полученный локальнооптимальный план, результаты проведенного в п. 6 исследования и текст программы.
9. Защитить лабораторную работу.

Технология Кобба-Дугласа. Ресурсов два, изменяются в пределах [1, 20]. Убывающая отдача от масштаба. Локально-A-оптимальное планирование.

|  |  |
| --- | --- |
| Критерий | Функционал |
|  |  |

1. **Описание алгоритма работы**

Пусть модель наблюдения описывается уравнением

где y – значение зависимой переменной; – вектор независимых переменных; – вектор неизвестных параметров; ошибка наблюдения, – нелинейная функция вектора параметров, в моделе Кобба-Дугласса выглядит следующим образом:

Информационная матрица Фишера для нелинейной модели:

*θtrue* – истинное значение вектора параметров.

В данной работе был выбран следующий вектор тэта:

1. **Ход работы**

Затравочная оценка:

0.19258063

0.46755664

0.45536742

Оптимальный план:

|  |  |
| --- | --- |
| X | Y |
| 1.3877551020408163  1.7755102040816326  4.1020408163265305  14.183673469387754  2.9387755102040813  19.612244897959183  4.489795918367347  9.142857142857142  19.612244897959183  20.0  1.7755102040816326  11.857142857142858  19.612244897959183  19.612244897959183  6.428571428571429 | 11.081632653061224  19.224489795918366  6.040816326530612  3.7142857142857144  7.591836734693877  17.6734693877551  14.571428571428571  3.326530612244898  3.326530612244898  16.51020408163265  19.612244897959183  20.0  3.7142857142857144  16.89795918367347  1.0 |

Среднее квадратичное отклонение при локально-А-оптимальном плане: 0.11221641551021046

Среднее квадратичное отклонение при случайном плане: 0.13488883754041595

1. **Выводы**

Таким образом, отклонение при локальнооптимальном плане меньше, чем при случайном плане, что говорит о его меньшей эффективности.

1. **Код программы**

**import** numpy **as** np  
**import** random  
**from** math **import** log, sqrt, e  
  
**def** f(x, y, theta):  
 **return** theta[0] \* x \*\* theta[1] \* y \*\* theta[2]  
  
**def** grad\_f(x, y, theta):  
 **return** np.array([[x \*\* theta[1] \* y \*\* theta[2]],  
 [theta[0] \* log(x) \* x \*\* theta[1] \* y \*\* theta[2]],  
 [theta[0] \* x \*\* theta[1] \* log(y) \* y \*\* theta[2]]  
 ]).reshape([3, 1])  
  
**def** getPlan(grid, N):  
 plan = []  
 **for** \_ **in** range(N):  
 plan.append([(random.choice(grid)), (random.choice(grid))])  
 **return** plan  
  
  
**def** getY(plan, theta, N, power):  
 Y = np.empty(shape=[N, 1])  
 **for** i **in** range(N):  
 Y[i] = f(plan[i][0], plan[i][1], theta)  
  
 sigma = sqrt(standard\_dev(Y) \* power)  
  
 **for** i **in** range(N): *# добавление шума в выход модели* Y[i] = Y[i] + random.normalvariate(0, sigma)  
 **return** Y  
  
**def** standard\_dev(Y):  
 mean = np.mean(Y)  
 diff = np.subtract(Y, mean)  
 **return** np.dot(diff.T, diff) / (Y.shape[0] - 1)  
  
**def** getX(plan):  
 N = len(plan)  
 X = np.empty(shape = [N, 3])  
 **for** i **in** range(N):  
 X[i] = [1.0, log(plan[i][0]), log(plan[i][1])]  
 **return** X  
  
**def** Least\_Squares(Y, X): *# метод наименьших квадратов* XtX\_1 = np.linalg.inv(np.dot(X.T, X))  
 result = np.dot(np.dot(XtX\_1, X.T), Y)  
 **return** result  
  
  
**def** M\_mat(x\_p, p, theta): *#Информационную матрицу M* dims = grad\_f(x\_p[0][0], x\_p[0][1], theta).shape[0]  
 M = np.zeros(shape = (dims, dims))  
 **for** i **in** range(dims):  
 **for** j **in** range(dims):  
 **for** k **in** range(len(x\_p)):  
 f = np.array(grad\_f(x\_p[k][0], x\_p[k][1], theta))  
 M[i][j] += p[k] \* f[i] \* f[j]  
 **return** M  
  
**def** find\_points(M, X, theta, flag): *# функция поиска экстремума* fi = [] *# массив значений экстремума* M1 = np.linalg.inv(M) *# получаем обратную матрицу от информационной матрицы M* M2 = np.linalg.matrix\_power(M1, 2) *# получаем обратную матрицу в квадрате (так как в формуле для поиска экстремума есть  
 # дифференциал от функцилнала критерия A по информационной матрице  
 # то, при вычислении дифференциала получаем обратную матрицу в степени 2)* **for** i **in** X: *# получаем набор чисел для поиска экстремума (в данном варианте - наибольнее число)* f\_i = np.array(grad\_f(i[0], i[1], theta)) *# получаем вектор-столбец от функции модели* buf = f\_i.T.dot(M2) *# 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* **if** flag == 0:  
 extr = max(fi)  
 **else**:  
 extr = min(fi)  
 **for** i **in** range(len(X)):  
 **if**(extr == fi[i]):  
 **return** i *# возвращаем наибольшее значение (экстремум) и точку экстремума***def** find\_point(x, X): *# проверяем принадлежность точки x массиву X* **for** i **in** X:  
 **if** x[0] == i[0] **and** x[1] == i[1]:  
 **return** 1  
 **return** 0  
  
**def** creat\_new\_multi(X, xs): *# создаём новый массив, исключая элементы массива xs* X\_new = []  
 **for** i **in** X:  
 **if** find\_point(i, xs) != 1:  
 X\_new.append(i)  
 **return** X\_new  
  
**def** funk(plan, plan\_w, theta): *# функция получения значения функционала от криетрия оптимальности* M = M\_mat(plan, plan\_w, theta)  
 M1 = np.linalg.inv(M) *# получаем обратную матрицу от информационной матрицы M* F = -np.trace(M1)  
 **return** (F)  
  
  
**def** building(plan, plan\_w, theta, X): *# функция построения оптимального плана* i = 0 *# счётчик успешных замен точек* X\_res = X.copy() *# копируем массив сетки* new\_plan = plan.copy() *# копируем точки плана* new\_plan\_i = new\_plan.copy() *# создаём следующую копию точек плана для полученния нового значения функционала* **while**(**True**):  
 M = M\_mat(new\_plan\_i, plan\_w, theta)  
 old\_funk = funk(new\_plan\_i, plan\_w, theta) *# получаем старое значение функционала* **if**(i == 0): *# если ноль замен, то обновляем план* new\_plan = new\_plan\_i.copy()  
  
 X\_new = creat\_new\_multi(X\_res, new\_plan\_i) *# получаем новую сетку, без точек плана* X\_max = find\_points(M, X\_new, theta, 0) *# максимум на множестве X \ plan (x\*)* X\_min = find\_points(M, new\_plan, theta, 1) *# минимум на множестве plan (x\*\*)* old\_plan = new\_plan\_i.copy() *# сохраняем старый план  
 # заменям точку x\*\* на точку x\** x\_star\_2 = new\_plan\_i[X\_min]  
 new\_plan\_i[X\_min] = X\_res[X\_max]  
  
 new\_funk = funk(new\_plan\_i, plan\_w, theta) *# получаем новый функционал* **if** (new\_funk > old\_funk): *# пункт 6 алгоритма* i += 1 *# увеличиваем счётких замен точек* del\_point = X\_res[X\_max] *# исключаем точку x\* из рассмотрения* X\_res.remove(del\_point)  
 **if** (x\_star\_2 **in** X\_res): *# исключаем точку x\*\* из рассмотрения* X\_res.remove(x\_star\_2)  
 **else**:  
 **if** (i == 0): *# больше ничего нельзя сделать, завершить вычисления (подпункт b пункта 6)* **return** new\_plan  
 **else**: *# возможно еще можно найти "удачные" замены (переход к пункту 2)* new\_plan\_i = old\_plan.copy()  
 i = 0  
  
**def** experiment(plan, exp\_n, N, error): *# функция эксперимента* dev\_rate = 0 *# переменная стандартного отклонения* th = np.array([[0.4], [0.3], [0.3]]) *# истинные значения тэта* **for** \_ **in** range(exp\_n): *# проводим несколько экспериментов* Y = getY(plan, theta, N, error) *# получаем выход из модели Кобба-Дугласа* Y = np.log(Y) *# приводим к логарифмическому виду* X = getX(plan) *# получаем план для оценки выхода Y* theta\_appr = Least\_Squares(Y, X) *# получаем оценку по методу наименьших квадратов* theta\_appr[0] = pow(e, theta\_appr[0])  
 theta\_err = np.subtract(th, theta\_appr)  
 dev\_rate += sqrt(np.dot(theta\_err.T, theta\_err))  
 dev\_rate /= exp\_n  
 **return** dev\_rate  
  
  
N = 15 *# число наблюдений*theta = [0.4, 0.3, 0.3]  
grid = np.linspace(1, 20) *# область изменения ресурсов для модели Кобба-Дугласа*plan = getPlan(grid, N) *# случайный план*X = getX(plan)  
Y = getY(plan, theta, N, 0.2)  
Y = np.log(Y)  
theta\_appr = Least\_Squares(Y, X)  
theta\_appr[0] = pow(e, theta\_appr[0])  
plan\_w = [1 / N] \* N  
  
print(**'затравочная оценка - '**, theta\_appr)  
  
X\_n = []  
  
**for** i **in** range(len(grid)):  
 **for** j **in** range(len(grid)):  
 X\_n.append((grid[i], grid[j]))  
  
optimal\_plan = building(plan, plan\_w, theta\_appr, X\_n)  
  
print(**"Оптимальный план "**)  
**for** i **in** optimal\_plan:  
 print(i)  
  
dev\_theta = experiment(optimal\_plan, 100, N, 0.2)  
print(**"Среднее квадратичное отклонение при локально-А-оптимальном плане "**, dev\_theta)  
  
dev\_theta = experiment(plan, 100, N, 0.2)  
print(**"Среднее квадратичное отклонение при случайном плане "**, dev\_theta)