# Министерство образования и науки Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»

NSTU_Logo_blue

## Кафедра прикладной математики

### Лабораторная работа № 1,2 по дисциплине «Статистические методы анализа данных»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| сигма градиент синий1 | Факультет: | ПМИ |  |  |
| Группа: | ПМИ-51 |  |  |
| Студент: | Тюшников В.В.  Кононов С.А. |  |  |
| Вариант: | 10 |  |  |
| Преподаватель: | Попов А.А |  |  |

Новосибирск

2018

**1. Постановка задачи**

1. В соответствии с вариантом задания выбрать имитационную модель объекта, диапазон изменения факторов, план эксперимента.

2. Написать программу по генерации экспериментальных данных. Полученные по программе данные оформить в виде одного или двух файлов унифицированной структуры, доступных для дальнейшей обработки. Построить графики зависимости отклика от входных факторов.

3. Спроектировать и сформировать программные модули по вычислению МНK-оценок параметров для заданной параметрической модели объекта. Предусмотреть достаточно простой способ настройки программы на необходимый вид (структуру) модели.

4. Пользуясь экспериментальными данными, полученными в лабораторной работе № 1, произвести оценку параметров модели объекта.

5. Произвести проверку адекватности полученной модели. В качестве можно взять величину дисперсии  , которая использовалась при зашумлении отклика в лабораторной работе № 1. Число степеней свободы 

**2. Вариант задания**

2) Произвести моделирование объекта, о котором известно: число действующих факторов – два; по первому фактору зависимость выхода близка к линейной (возрастающей), по второму – существенно нелинейная. Максимум выходной величины приходится на граничные точки области действия факторов.

**3. Моделирование**

1. Имитационная модель представлена в виде функции:

, 

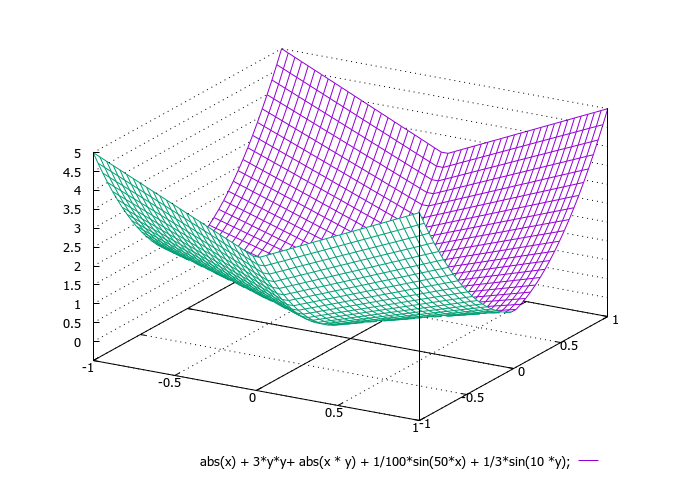
2. Область действия факторов:



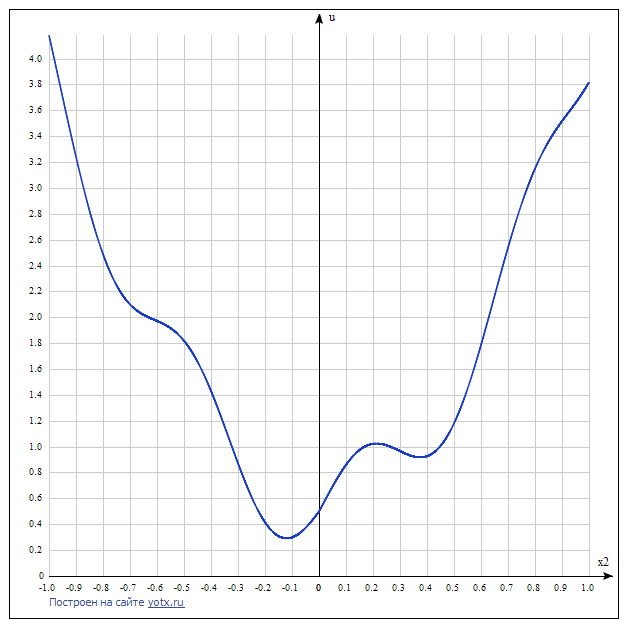
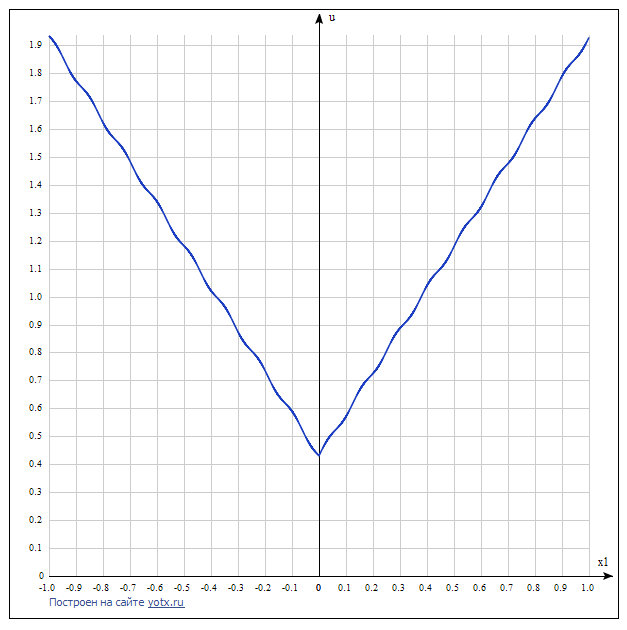
3. Количество точек проведения наблюдения – 16. Распределение точек – равномерное на двумерной сетке с шагом 0.5 по обеим переменным

4. Вычисление отклика  , где e – реализация случайной величины (Нормальное распределение с )

Общий вид функции  :



u(x1, 0.5) u(0.5, x2)



**4. Оценивание параметров модели**

1. Будем вычислять оценку  при помощи метода наименьших квадратов, который выглядит как:



Для нашей задачи метод выглядит как:

, где



или, что то же самое, - решение СЛАУ:





2. Оценку  будем вычислять как:





3. Проверим гипотезу об адекватности модели вида:



Гипотеза отвергается, если 

В нашем случае при :



А значит, гипотеза не отвергается, можно считать, что модель адекватна.

**Приложение**

**Текст программы**

import numpy as np

import random as rnd

def getFuncIX(xVec, i):

switcher = {

0: abs(xVec[0]),

1: xVec[1] \* xVec[1],

2: abs(xVec[0] \* xVec[1]),

3: np.sin(xVec[0] \* 50),

4: np.sin(xVec[1] \* 10)

}

return switcher[i]

def getU(xVec, tetaVec):

summ = 0

for i in range(tetaVec.\_\_len\_\_()):

summ += getFuncIX(xVec, i) \* tetaVec[i]

return summ

def getY(xVec, tetaVec, mu, sigma):

return getU(xVec, tetaVec) + rnd.normalvariate(mu, sigma)

def makeObservations(bordersX, steps, tetaVec, sigma, mu):

obsTable = []

for x1 in np.arange(bordersX[0][0], bordersX[0][1] + steps[0], steps[0]):

for x2 in np.arange(bordersX[1][0], bordersX[1][1] + steps[1], steps[1]):

yDash = getY([x1, x2], tetaVec, mu, sigma)

obsTable.append([x1, x2, yDash])

return obsTable

def avg(listOfFloat):

avg = 0

n = listOfFloat.\_\_len\_\_()

for i in range(n):

avg += listOfFloat[i]

avg /= n

return avg

def calcSignalPower(tetaVec):

uVec = []

for x1 in np.arange(-1, 1 + 0.125, 0.125):

for x2 in np.arange(-1, 1 + 0.125, 0.125):

y = getU([x1, x2], tetaVec)

uVec.append(y)

uAvg = avg(uVec)

omega2 = 0

n = uVec.\_\_len\_\_()

for i in range(n):

omega2 += (uVec[i] - uAvg) \* (uVec[i] - uAvg)

omega2 /= (n - 1)

return omega2

def calcTetaDashVec(obsTable, tetaVeclen):

# Solving system (XT \* X) \* TetaDash = XT \* Y

# for TetaDash

m = tetaVeclen

n = obsTable.\_\_len\_\_()

X = np.zeros((n, m), dtype=float)

Y = np.zeros((n, 1), dtype=float)

for i in range(n):

x1 = obsTable[i][0]

x2 = obsTable[i][1]

y = obsTable[i][2]

Y[i] = y

for j in range(m):

X[i][j] = getFuncIX([x1, x2], j)

XT = X.transpose()

XTX = np.dot(XT, X)

XTY = np.dot(XT, Y)

TetaDash = np.linalg.solve(XTX, XTY)

return np.transpose(TetaDash)[0]

def calcSigma2Dash(tetaDashVec, obsTable):

m = tetaDashVec.\_\_len\_\_()

n = obsTable.\_\_len\_\_()

Y = np.zeros((n, 1), dtype=float)

YDash = np.zeros((n, 1), dtype=float)

for i in range(n):

x1 = obsTable[i][0]

x2 = obsTable[i][1]

y = obsTable[i][2]

Y[i] = y

YDash[i] = getU([x1, x2], tetaDashVec)

eDash = np.subtract(Y, YDash)

sigma2 = np.dot(np.transpose(eDash), eDash)

sigma2 /= (n - m)

return sigma2[0][0]

def calcQuantile(sigma, sigmaDash):

print("F: " + str(sigmaDash/sigma))

if sigmaDash/sigma <= 1.28:

return "Модель адекватная"

else:

return "Модель неадекватная"

sigma = np.sqrt(1.1 \* 0.1)

mu = 0

bordersX = [[-1, 1], [-1, 1]]

tetaVec = [1, 3,1, 1 / 100, 1 / 3]

observations = []

observations = makeObservations(bordersX, [0.5, 0.5], tetaVec, sigma, mu)

for i in range(observations.\_\_len\_\_()):

print(observations[i])

print("Power: ", calcSignalPower(tetaVec))

print("Teta: ", tetaVec)

tetaDashVec = calcTetaDashVec(observations, tetaVec.\_\_len\_\_())

print("TetaDash: ", tetaDashVec)

sigma2Dash = calcSigma2Dash(tetaDashVec, observations)

print("sigma2: ", sigma \* sigma)

print("sigma2Dash: ", sigma2Dash)

print(calcQuantile(sigma\*sigma, sigma2Dash))