Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчет

по лабораторной работе №3

Линейные модели

Выполнил:

студент гр. ИП-712

Алексеев С.В.

Проверил:

Ассистент кафедры  
Морозова К.И.

Новосибирск, 2020 г.

Оглавление

[Теоретический базис 3](#_Toc25778032)

[Результаты 6](#_Toc25778033)

[Код программы 8](#_Toc25778034)

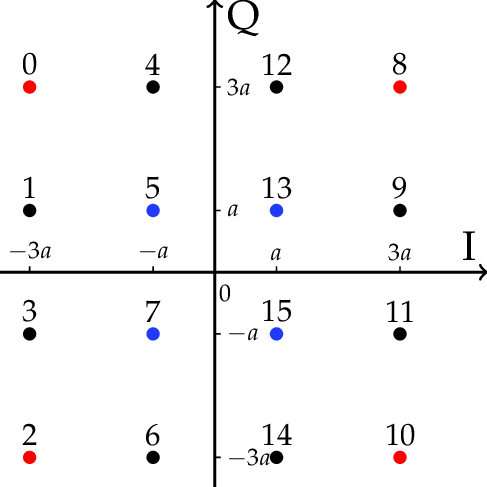
# Теоретический базис

Задача состоит в том, чтобы компенсировать нелинейные искажения, возникающие при передаче данных по волоконно-оптическим каналам связи.

Главные аспекты системы:

1. каждый передаваемый импульс задаётся комплексным числом (варианты чисел определяются модуляцией 16-QAM);
2. всего передаётся одновременно 6 потоков импульсов: 3 канала, в каждом канале 2 поляризации.

Рассмотрим модуляцию 16-QAM. Согласно этой модуляции каждый импульс описывается двумя числами (или, иначе, одним комплексным числом). Если рассмотреть эти числа как координаты, то сетка допустимых значений будет следующей:



В данных, прилагаемых к лабораторной работе значение . Если все значения разделить на указанное значение , то получатся числа: -3,-1,1,3.

Нелинейные искажения в оптических каналах связи имеют сложную структуру, но в рамках данной лабораторной работы мы будем вести речь исключительно о межимпульсном взаимодействии. Суть его заключается в том, что в процессе перемещения по каналу связи, импульс «расплывается» и начинает влиять на значения последующих и предыдущих импульсов. Чем дальше находятся друг от друга импульсы, тем меньше их влияние. Рассмотрим самую простую линейную модель, будем считать, что влияние соседнего импульса можно учесть, если взять его значение и умножить на комплексный коэффициент, тогда:

где – число рассматриваемых соседей как слева, так и справа, – значение переданного импульса (т.е. до внесения в него искажений), – значение импульса на приёмнике (т.е. с искажением).

Так как речь идёт о комплексных числах, а рассматриваемая в рамках лабораторной работы библиотека умеет работать только с вещественными числами, то необходимо расписать каждое уравнение импульса на 2. Покажем на примере с 1 соседом:

Опишем каждое комплексное число в виде двух составляющих:

.

Раскроем скобки:

Сгруппируем по действительной и мнимой частям:

Теперь можно рассмотреть эти части по отдельности и получить тем самым 2 уравнения:

Таким образом, мы можем построить для каждого импульса 2 уравнения и получить максимум (где n – число соседей) уравнений с неизвестными коэффициентами. Необходимо подобрать такое количество уравнений, необходимое для обучения и такое количество соседей, чтобы количество ошибок стало минимальным.

Для определения ошибки необходимо взять значение, полученное на приёмнике, и определить ближайшее к нему исходное на сетке 16-QAM (то есть одно из 16 значений). Чтобы проверить рассматриваемый в лабораторной работе метод, необходимо сначала посчитать количество ошибок без учёта межимпульсного взаимодействия, а потом преобразовать значения на приёмнике при помощи формулы, указанной выше, после чего опять посчитать число ошибок и сравнить полученные результаты.

# Результаты

В качестве модели была выбрана линейная регрессия.

Программа перебирает отношение обучающей выборки к общему количеству данных от 0.1 до 0.6 с шагом 0.1, а также количество соседних импульсов (с каждой из сторон) от 1 до 9 с тем, чтобы найти такое соотношение, при котором получается наименьшее количество ошибок.

Результат запуска программы:

Original: 0.006496429443359375

Ratio: 0.1

Neighbours: 1 Train errors: 0.007708746756220425 Test errors: 0.006832708838439497

Neighbours: 2 Train errors: 0.0075584058634906096 Test errors: 0.0068117465538581525

Neighbours: 3 Train errors: 0.007522529402779899 Test errors: 0.0067992607287586685

Neighbours: 4 Train errors: 0.007563025210084034 Test errors: 0.006846121237812633

Neighbours: 5 Train errors: 0.007527128228641296 Test errors: 0.0068505926440913635

Neighbours: 6 Train errors: 0.00752942975080263 Test errors: 0.00680419189092944

Neighbours: 7 Train errors: 0.00764642911760208 Test errors: 0.006851057335210025

Neighbours: 8 Train errors: 0.007725256233746367 Test errors: 0.0068004137907643255

Neighbours: 9 Train errors: 0.007727620504973221 Test errors: 0.006783685237005003

Ratio: 0.2

Neighbours: 1 Train errors: 0.006619610835558947 Test errors: 0.006909607644915787

Neighbours: 2 Train errors: 0.006582462031595818 Test errors: 0.006895565092989986

Neighbours: 3 Train errors: 0.006583466910922831 Test errors: 0.006867214772141999

Neighbours: 4 Train errors: 0.006603557523475074 Test errors: 0.00686270769348162

Neighbours: 5 Train errors: 0.006566389249446438 Test errors: 0.006820046166466358

Neighbours: 6 Train errors: 0.006548300878197785 Test errors: 0.006820306389148559

Neighbours: 7 Train errors: 0.006644772015580845 Test errors: 0.006873032528856244

Neighbours: 8 Train errors: 0.006588495913222825 Test errors: 0.006882834411310171

Neighbours: 9 Train errors: 0.0067041026816410726 Test errors: 0.006859247104615444

Ratio: 0.30000000000000004

Neighbours: 1 Train errors: 0.00707091260555499 Test errors: 0.00675219078345032

Neighbours: 2 Train errors: 0.007071632071632071 Test errors: 0.006725235437739798

Neighbours: 3 Train errors: 0.00705963162714969 Test errors: 0.006725528667974711

Neighbours: 4 Train errors: 0.007073071443110116 Test errors: 0.00674762361559257

Neighbours: 5 Train errors: 0.007086513994910942 Test errors: 0.006742467187023067

Neighbours: 6 Train errors: 0.007099959283387622 Test errors: 0.006693703122274551

Neighbours: 7 Train errors: 0.007113407309375954 Test errors: 0.006715799572630936

Neighbours: 8 Train errors: 0.0071650376705355324 Test errors: 0.006737897950283471

Neighbours: 9 Train errors: 0.007191222889726097 Test errors: 0.006678224083038946

Ratio: 0.4

Neighbours: 1 Train errors: 0.006590492904013429 Test errors: 0.007089267548321465

Neighbours: 2 Train errors: 0.006667302556276231 Test errors: 0.007076911338318327

Neighbours: 3 Train errors: 0.006639194139194139 Test errors: 0.007083630074270018

Neighbours: 4 Train errors: 0.006611081431733191 Test errors: 0.007045836088925065

Neighbours: 5 Train errors: 0.006640207601892841 Test errors: 0.007046194546194547

Neighbours: 6 Train errors: 0.006573925654530188 Test errors: 0.007052912744848639

Neighbours: 7 Train errors: 0.00659351145038168 Test errors: 0.007046911570163834

Neighbours: 8 Train errors: 0.006594014810290862 Test errors: 0.007028189080547499

Neighbours: 9 Train errors: 0.006613605130554283 Test errors: 0.007073071443110116

Ratio: 0.5

Neighbours: 1 Train errors: 0.006782945736434108 Test errors: 0.0069355429408533235

Neighbours: 2 Train errors: 0.006806250763032597 Test errors: 0.006958857282383104

Neighbours: 3 Train errors: 0.006821927843233014 Test errors: 0.006959282095110189

Neighbours: 4 Train errors: 0.006822344322344322 Test errors: 0.00695970695970696

Neighbours: 5 Train errors: 0.006799865681665547 Test errors: 0.006998290493925148

Neighbours: 6 Train errors: 0.006853706191232141 Test errors: 0.007006349981682745

Neighbours: 7 Train errors: 0.006846492031507602 Test errors: 0.006991512487024486

Neighbours: 8 Train errors: 0.006801111382510992 Test errors: 0.0069766731802638005

Neighbours: 9 Train errors: 0.00683969465648855 Test errors: 0.006977099236641221

Ratio: 0.6

Neighbours: 1 Train errors: 0.006841476131132531 Test errors: 0.0068477473009575405

Neighbours: 2 Train errors: 0.006816389856804945 Test errors: 0.006915035672046087

Neighbours: 3 Train errors: 0.006816736614523719 Test errors: 0.006886947231866916

Neighbours: 4 Train errors: 0.0068615979447002266 Test errors: 0.00689701224863586

Neighbours: 5 Train errors: 0.006823789778942281 Test errors: 0.006897538637664568

Neighbours: 6 Train errors: 0.006824136966089191 Test errors: 0.006878983322520322

Neighbours: 7 Train errors: 0.006824484188566922 Test errors: 0.006889050036258158

Neighbours: 8 Train errors: 0.006786668362803714 Test errors: 0.00691820298484675

Neighbours: 9 Train errors: 0.0067806528763707605 Test errors: 0.0068996449975188

Min error 0.006467244076462284 - with 3 neighbours and 60.0 % train data

Better than original by 0.43925242629894413 %

Значения ошибок – это отношение количества некорректных битов к общему количеству переданных битов.

Наилучший результат получился при отношении 0.6 размера обучающей выборки к общему количеству данных и 3 соседних импульсах с каждой стороны (то есть суммарно учитывалось 7 импульсов). Уменьшение ошибок составило 0.44% по отношению к изначальному количеству ошибок.

# Код программы

import math

import random

import numpy as np

import sklearn.linear\_model

def load\_data(filename):

    with open(filename, 'rb') as file:

        n\_channels = np.fromfile(file, dtype=np.int32, count=1)[0]

        n\_samples = np.fromfile(file, dtype=np.int32, count=1)[0]

        n\_modes = np.fromfile(file, dtype=np.int32, count=1)[0]

        flags = np.fromfile(file, dtype=np.int32, count=1)[0]

        data = np.fromfile(file, dtype=np.complex)

        data = data.reshape((n\_channels \* n\_modes, n\_samples), order='F')

        data = data[2]

        return data

def normalize(data, minimum):

    length = data.shape[0]

    for i in range(0, length):

        data[i] /= minimum

def find\_min(data):

    m = 9999999

    for d in data:

        m = min(m, abs(np.real(d)), abs(np.imag(d)))

    return m

def qam\_value(q, i):

    result = 0

    if i < -2:

        result = 0

    elif i < 0:

        result = 4

    elif i < 2:

        result = 12

    else:

        result = 8

    if q < -2:

        result += 2

    elif q < 0:

        result += 3

    elif q < 2:

        result += 1

    return result

def count\_set\_bits(n):

    count = 0

    while (n):

        count += n & 1

        n >>= 1

    return count

def get\_original\_errors\_percentage(rx, tx):

    length = rx.shape[0]

    n\_errors = 0

    for i in range(length):

        r\_value = qam\_value(np.real(rx[i]), np.imag(rx[i]))

        t\_value = qam\_value(np.real(tx[i]), np.imag(tx[i]))

        n\_errors += count\_set\_bits(r\_value ^ t\_value)

        # if r\_value != t\_value:

        #     n\_errors += 1

    return n\_errors / (length\*4)

def create\_regression(rx\_real, rx\_imag, tx\_real, tx\_imag, one\_side\_neighbours):

    x = construct\_xs(rx\_real, rx\_imag, one\_side\_neighbours)

    y = construct\_ys(tx\_real, tx\_imag, one\_side\_neighbours)

    regression = sklearn.linear\_model.LinearRegression()

    regression.fit(x, y)

    return regression

def construct\_xs(rx\_real, rx\_imag, one\_side\_neighbours):

    neigh = one\_side\_neighbours

    x\_num = 2 \* (rx\_real.shape[0] - 2 \* neigh)

    x = np.empty((x\_num, 2 \* (1 + 2 \* neigh)))

    j = 0

    for i in range(neigh, neigh + int(x\_num / 2)):

        reals = rx\_real[i - neigh : i + neigh + 1]

        imags = rx\_imag[i - neigh : i + neigh + 1]

        x[j] = np.concatenate((reals, np.negative(imags)), axis=0)

        j += 1

        x[j] = np.concatenate((imags, reals), axis=0)

        j += 1

    return x

def construct\_ys(tx\_real, tx\_imag, one\_side\_neighbours):

    neigh = one\_side\_neighbours

    length = tx\_real.shape[0]

    y\_real = tx\_real[neigh : length - neigh]

    y\_imag = tx\_imag[neigh : length - neigh]

    # interleaving the Re and Im parts: y\_real[0], y\_imag[0], y\_real[1], ...

    y = np.ravel(np.column\_stack((y\_real, y\_imag)))

    return y

def calc\_errors\_percentage(rx\_train, rx\_test, tx\_train, tx\_test, one\_side\_neighbours):

    regression = create\_regression(

        np.real(rx\_train), np.imag(rx\_train),

        np.real(tx\_train), np.imag(tx\_train),

        one\_side\_neighbours

    )

    def errors\_percentage(rx, tx, neigh):

        x = construct\_xs(np.real(rx), np.imag(rx), neigh)

        y = construct\_ys(np.real(tx), np.imag(tx), neigh)

        predicted\_y = regression.predict(x)

        length = int(x.shape[0] / 2)

        errors = 0

        for i in range(length):

            test\_value = qam\_value(y[i \* 2], y[i \* 2 + 1])

            predicted\_value = qam\_value(predicted\_y[i \* 2], predicted\_y[i \* 2 + 1])

            errors += count\_set\_bits(test\_value ^ predicted\_value)

            # if test\_value != predicted\_value:

            #     errors += 1

        return errors / (length \* 4)

    train\_err = errors\_percentage(rx\_train, tx\_train, one\_side\_neighbours)

    test\_err = errors\_percentage(rx\_test, tx\_test, one\_side\_neighbours)

    return train\_err, test\_err

def split\_data(data, ratio):

    length = data.shape[0]

    n\_train = int(length \* ratio)

    return data[:n\_train], data[n\_train:]

def unison\_shuffled\_copies(a, b):

    assert len(a) == len(b)

    p = np.random.permutation(len(a))

    return a[p], b[p]

rx = load\_data('rx.bin')

tx = load\_data('tx.bin')

minimum = find\_min(tx)

normalize(rx, minimum)

normalize(tx, minimum)

original\_error = get\_original\_errors\_percentage(rx, tx)

print("Original: ", original\_error)

min\_error = original\_error

min\_error\_neighbours = 0

min\_error\_ratio = 0

ratios = np.arange(0.1, 0.7, 0.1)

for ratio in ratios:

    rx, tx = unison\_shuffled\_copies(rx, tx)

    rx\_train, rx\_test = split\_data(rx, ratio)

    tx\_train, tx\_test = split\_data(tx, ratio)

    print("Ratio:", ratio)

    for neigh in range(1, 10):

        train\_err, test\_err = calc\_errors\_percentage(rx\_train, rx\_test, tx\_train, tx\_test, neigh)

        if test\_err < min\_error:

            min\_error = test\_err

            min\_error\_neighbours = neigh

            min\_error\_ratio = ratio

        print("\tNeighbours:", neigh, "Train errors:", train\_err, "Test errors: ", test\_err)

print("Min error", min\_error, "- with", min\_error\_neighbours,

    "neighbours and ", min\_error\_ratio\*100, "% train data")

print("Better than original by", (original\_error - min\_error) / original\_error \* 100, "%")