Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Томский государственный университет систем управления и радиоэлектроники

(ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**ПОМЕХОЗАЩИЩАЮЩИЕ КОДЫ И МЕХАНИЗМЫ ИЗБЫТОЧНОСТИ**

ОТЧЕТ

К лабораторной работе №3 по дисциплине “Сети и телекоммуникации”

Студенты гр. 589-3

Бураков И. С.

Потлог А.М.

Слесаренко А.С.

Хромов В.А.

19.10.2022

Принял:

к.т.н., доцент каф. КСУП

Коколов А.А.

Томск 2022

Оглавление

[**1 Введение** 3](#_Toc116926542)

[**2 Задание** 4](#_Toc116926543)

[**3 Фазовая манипуляция** 5](#_Toc116926544)

[**4 Частотная манипуляция** 6](#_Toc116926545)

[**5 Построение фазового приемника** 7](#_Toc116926546)

[**6 Построение частотного приемника** 10](#_Toc116926547)

[**7 Заключение** 12](#_Toc116926548)

[**Приложение А** 13](#_Toc116926549)

# **1 Введение**

Обнаружение ошибок в технике связи — действие, направленное на контроль целостности данных при записи/воспроизведении информации или при её передаче по линиям связи. Исправление ошибок (коррекция ошибок) — процедура восстановления информации после чтения её из устройства хранения или канала связи.

Для обнаружения ошибок используют коды обнаружения ошибок, для исправления — корректирующие коды (коды, исправляющие ошибки, коды с коррекцией ошибок, помехоустойчивые коды).

# **2 Задание**

1. Повторить все графики приведённые в данном методическом пособии для своего случайного сигнала (8 графиков). Знать алгоритмы всех используемых функций. Уметь сделать выводы по каждому графику;

2. Самостоятельно изучить как минимум 2 других алгоритма помехоустойчивого кодирования;

3. Оформить отчет.

# **3 Цепной код**

Цепной код является самым простым из всех известных сверточных кодов, которые относятся к непрерывным или рекуррентным кодам. В цепном коде каждый проверочный элемент формируется путём сложения двух информационных элементов, отстоящих один от другого на t элементов, по модулю.

Поскольку каждый информационный элемент участвует в формировании двух проверочных элементов, а каждый проверочный элемент формируется по двум информационным, число проверочных элементов, сформированных за время t, будет равно числу информационных элементов, поступивших за то же время на вход кодирующего устройства.

Следовательно, избыточность цепного кода равна 1/ 2.

В канал связи передаётся последовательность импульсов, в которой за каждым информационным следует проверочный.

При декодировании принимаемая последовательность единичных элементов делится на две независимые последовательности: информационную и проверочную. Из информационной последовательности по тому же принципу, что и при кодировании, снова формируется проверочная последовательность. Единичные элементы вновь полученной проверочной последовательности сравниваются с единичными элементами принятой проверочной последовательности. Если соответствующие сравниваемые элементы совпадают, считается, что ошибки нет, в противном случае должно быть принято решение об ошибке.

Для исправления ошибочно принятого информационного элемента необходимо, чтобы предыдущий и последующий информационные элементы этой цепи, а также соответствующие проверочные элементы были приняты правильно.

Это условие выполняется полностью, если длина пакета ошибок не превышает величины 2t (t информационных и t проверочных, которые передаются в канал связи с задержкой 2t +1).

При этом необходимо, чтобы между последним элементом данного пакета ошибок и первым элементом последующего пакета ошибок было принято подряд не менее 4t неискаженных элементов.

Несмотря на свою простоту, цепной код позволяет исправлять одиночные и групповые ошибки.

# **4 Применение цепного кода на практике**

Пусть мы имеем последовательность 128 битов информации, сформированной случайным образом.

N = 128

k = 8

i = range(0,N)

x = []

for index in i:

x.append(math.floor(random.uniform(0, 2)))

где x — исходный случайный сигнал,

N — кол-во точек,

K — шаг сложения.

Сформированный сигнал показан на рисунке 4.1.

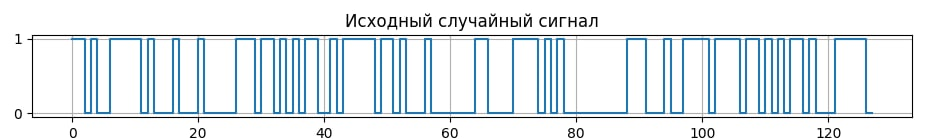


Рисунок 4.1 – Исходный случайный сигнал

Теперь формируется код, в котором после каждой информационной посылки следует проверочная, которая формируется путём сложения по модулю двух информационных посылок, отстоящих одна от другой на шаг сложения k, в данном случае k=8.

def code(x,k):

s = []

count = 0

for i in range(len(x)):

if i + k >= len(x):

s.append(x[i])

s.append(int(bool(x[i]) != bool(x[count])))

count = count + 1

else:

s.append(x[i])

s.append(int(bool(x[i]) != bool(x[i+k])))

return s

Сформированный сигнал, содержащий как информационные, так и проверочные посылки показан на рисунке 4.2.



Рисунок 4.2 – Сформированный сигнал

Применим над исходным сигналом помехи.

def nois(x):

MIN = 2

MAX = 5

xres = x.copy()

for i in range(MIN + math.floor(random.uniform(0, int(MAX - MIN + 1)))):

ri = math.floor(random.uniform(0,len(x)))

xres[ri] = int(not bool(x[ri]))

return xres

Для наглядности на рисунке 4.3 изображена разница между переданным и принятым сигналом, на рисунке 4.4 изображена разность между исходными и выделенными из закодированного сигнала с ошибками информационными посылками, на рис. 4.5 разность проверочных посылок, сформированных при кодировании и выделенной посылки из сигнала с ошибками.

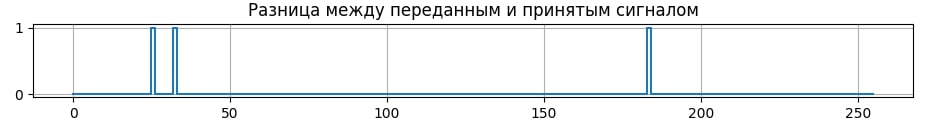


Рисунок 4.3 – Разница между переданным и принятым сигналом



Рисунок 4.4 – Разница между информационными посылками

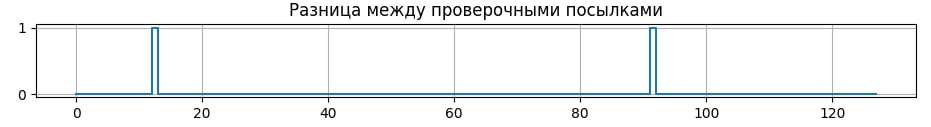


Рисунок 4.5 – Разница между проверочными посылками

При приёме информационные и проверочные посылки разделяются и регистрируются независимо друг от друга. Затем из принятой последовательности информационных посылок формируются контрольные, как при передаче, которая затем сравнивается с принятой и проверочной.

На рисунке 4.6 — разность между контрольной и проверочной посылками. Причём, если ошибка была в информационном разряде, то она выглядит продублированной, т.е. двойной на расстоянии k=8. Если ошибка возникла в проверочных разрядах, то она не дублируется и затем просто игнорируется.



Рисунок 4.6 – Разность между контрольной и проверочной посылкой

Исправляем i-й разряд в принятой информационной последовательности, если у нас в разности, изображённой на рисунке 3.8, на i-м и (i+k)-м месте присутствует импульс и т. д.

def deCode(x,dif,k):

count = 0

for i in range(len(x)):

if i + k >= len(x):

if dif[i] == 1 and dif[count] == 1:

x[count] = int(not bool(x[count]))

count = count + 1

else:

if dif[i] == 1 and dif[i+k] == 1:

x[i+k] = int(not bool(x[i+k]))

return x

В итоге получаем первоначальную информационную посылку без ошибок (рисунок 4.7) из сигнала, принятого с ошибками. Для наглядности на рисунке 4.8 изображена разница между исходной и восстановленной информационной посылкой.

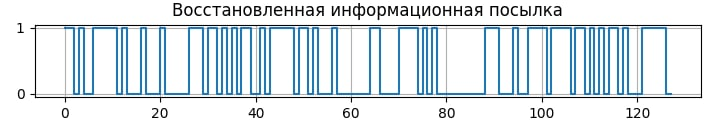


Рисунок 4.7 – Восстановленная информационная посылка



Рисунок 4.8 – Разница между исходной и восстановленной информационной посылкой

# **7 Заключение**

В ходе выполнения лабораторной работы были получены навыки построения помехозащищеного сигнала.

# **Приложение А**

Листинг программы

Файл lab3.py

import matplotlib.pyplot as plt

import math

import numpy

import random

import CodeLab

import Nois

import DifferenceBoolList

import GetPackage

N = 128

k = 8

i = range(0,N)

x = []

for index in i:

x.append(math.floor(random.uniform(0, 2)))

code = CodeLab.code(x,k)

plt.figure()

plt.subplot(511)

plt.step([i for i in range(N)], x, where='post')

plt.title('Исходный случайный сигнал')

plt.grid(True)

plt.subplot(512)

plt.step([i for i in range(len(code))], code, where='post')

plt.title('Сформированный сигнал')

plt.grid(True)

#Сформированый сигнал с помехами

codeNois = Nois.nois(code)

dif = DifferenceBoolList.difference(code, codeNois)

plt.subplot(513)

plt.step([i for i in range(len(dif))], dif, where='post')

plt.title('Разница между переданным и принятым сигналом')

plt.grid(True)

dif = DifferenceBoolList.difference(GetPackage.getInformationPackage(code),

GetPackage.getInformationPackage(codeNois))

plt.subplot(514)

plt.step([i for i in range(len(dif))], dif, where='post')

plt.title('Разница между информационными посылками')

plt.grid(True)

dif = DifferenceBoolList.difference(GetPackage.getVerificationPackage(code),

GetPackage.getVerificationPackage(codeNois))

plt.subplot(515)

plt.step([i for i in range(len(dif))], dif, where='post')

plt.title('Разница между проверочными посылками')

plt.grid(True)

#plt.show()

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

plt.figure()

codeReceived = CodeLab.code(GetPackage.getInformationPackage(codeNois), k)

dif = DifferenceBoolList.differenceRec(GetPackage.getVerificationPackage(codeNois),

GetPackage.getVerificationPackage(codeReceived))

plt.subplot(511)

plt.step([i for i in range(len(dif))], dif, where='post')

plt.title('Разность между контрольной и проверочной посылкой')

plt.grid(True)

decode = CodeLab.deCode(GetPackage.getInformationPackage(codeNois),

dif,

k)

plt.subplot(512)

plt.step([i for i in range(len(decode))], decode, where='post')

plt.title('Восстановленная информационная посылка')

plt.grid(True)

dif = DifferenceBoolList.difference(decode, x)

plt.subplot(513)

plt.step([i for i in range(len(dif))], dif, where='post')

plt.title('Разница между исходной и восстановленной информационной посылкой')

plt.grid(True)

plt.show()

Файл GetPackage.py

def getInformationPackage(s):

result = []

for i in range(len(s)):

if i % 2 == 0:

result.append(s[i])

return result

def getVerificationPackage(s):

result = []

for i in range(len(s)):

if i % 2 == 1:

result.append(s[i])

return result

Файл CodeLab.py

import DifferenceBoolList

def code(x,k):

s = []

count = 0

for i in range(len(x)):

if i + k >= len(x):

s.append(x[i])

s.append(int(bool(x[i]) != bool(x[count])))

count = count + 1

else:

s.append(x[i])

s.append(int(bool(x[i]) != bool(x[i+k])))

return s

def deCode(x,dif,k):

count = 0

for i in range(len(x)):

if i + k >= len(x):

if dif[i] == 1 and dif[count] == 1:

x[count] = int(not bool(x[count]))

count = count + 1

else:

if dif[i] == 1 and dif[i+k] == 1:

x[i+k] = int(not bool(x[i+k]))

return x

Файл Nois.py

import math

import random

def nois(x):

MIN = 2

MAX = 5

xres = x.copy()

for i in range(MIN + math.floor(random.uniform(0, int(MAX - MIN + 1)))):

ri = math.floor(random.uniform(0,len(x)))

xres[ri] = int(not bool(x[ri]))

return xres

Файл DifferenceBoolList.py

import math

def difference(list1, list2):

result = []

for i in range(len(list1)):

result.append(int(bool(list1[i]) != bool(list2[i])))

return result

def differenceRec(list1, list2):

result = []

for i in range(len(list1)):

result.append(abs(list1[i] - list2[i]))

return result