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)

**МНОГОКАНАЛЬНЫЕ СИСТЕМЫ ПЕРЕДАЧИ ДАННЫХ**

ОТЧЕТ

К лабораторной работе №4 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**

Любой сигнал занимает определённую полосу частот, существует некоторое время, обладает ограниченной энергией и распространяется в определённой области пространства. В соответствии с этим выделяют четыре вида ресурса канала: частотный, временной, энергетический и пространственный.

При частотном разделении каналов для передачи данных различных источников сообщений используются определенные поддиапазоны частот.

Временное разделение основано на возможности передачи вместо непрерывных сигналов последовательных импульсов (отсчётов). Поскольку при импульсной передаче период следования импульсов обычно намного больше их длительности (импульсы имеют большую скважность), между импульсами одного сигнала остаётся промежуток, на котором можно разместить импульсы от других сигналов. В настоящее время уже реализованы многоканальные системы с временным разделением 12, 15, 30, 120, 480 речевых сигналов.

# **2 Задание**

1. Варианты исходных данных представлены в приложении В;

2. Повторить все графики, приведённые в данном методическом пособии

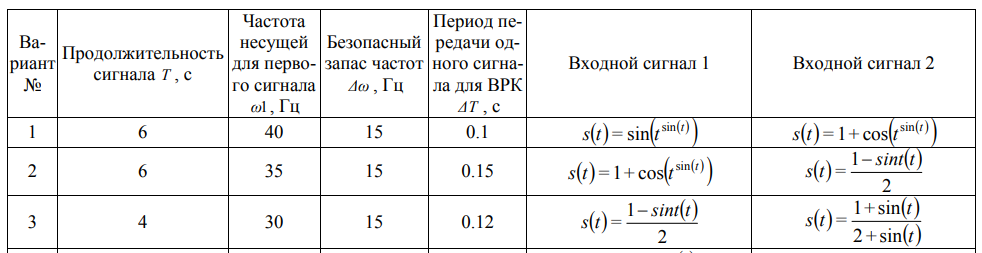
для своего варианта (22 графика).

3. Изучить алгоритмы всех используемых функций.

4. Сделать выводы по каждому графику.

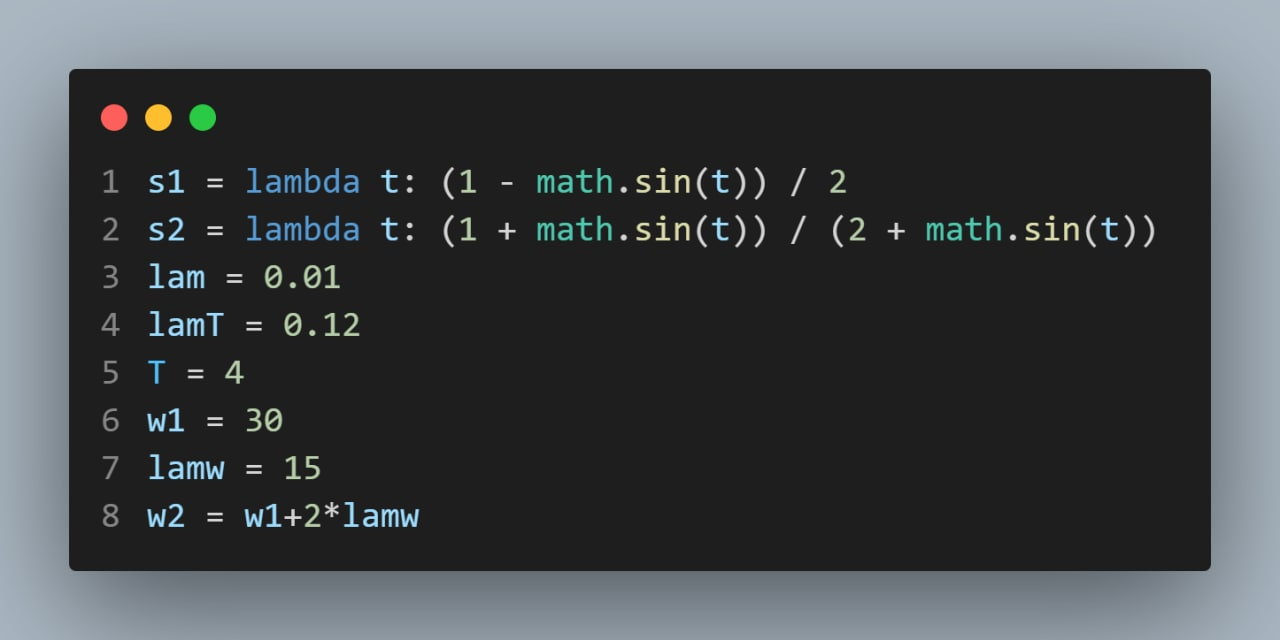
5. Оформить отчет.

Вариант 3



# **3 Принцип частотного объединения и разделения каналов на практике**

Моделирование будет проходить со следующими начальными параметрами:



где s1(t), s2(t) — исходные сигналы;

∆ — элементарный квант времени;

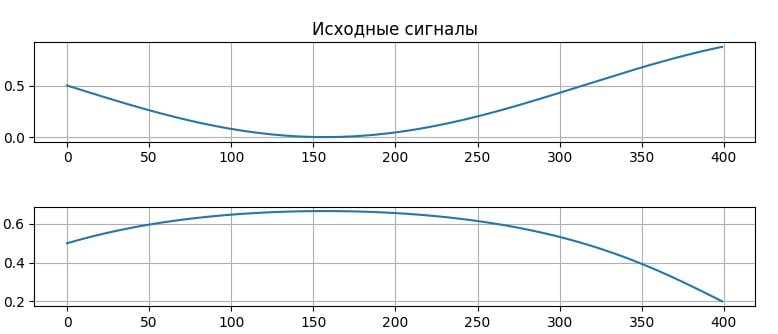
∆T — период передачи одного сигнала для ВРК;

T — время моделирования;

ω1 — частота несущей для первого сигнала

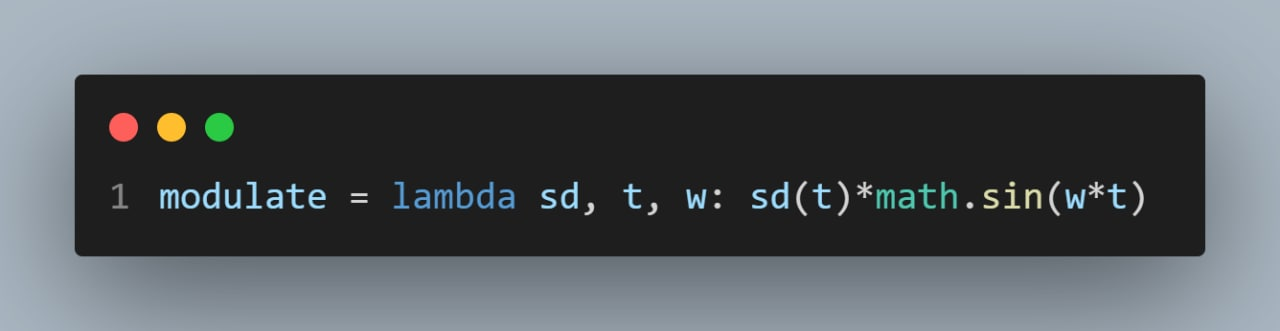
∆ω — безопасный запас частот.

Исходные сигналы изображены на рис. 4.7.



Прежде чем передать сигналы, их необходимо модулировать на разные

несущие частоты. Модуляция осуществляется с помощью формулы

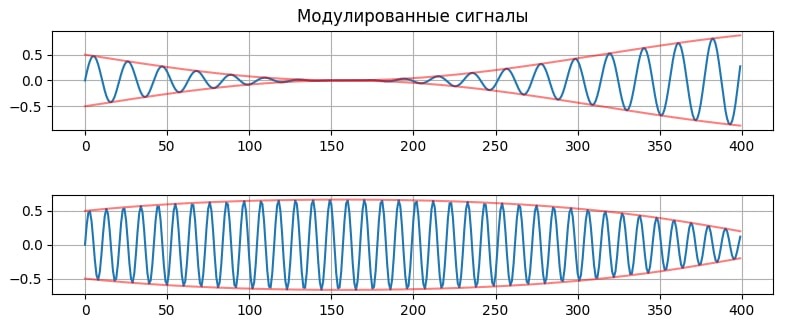


где sd — входной сигнал;

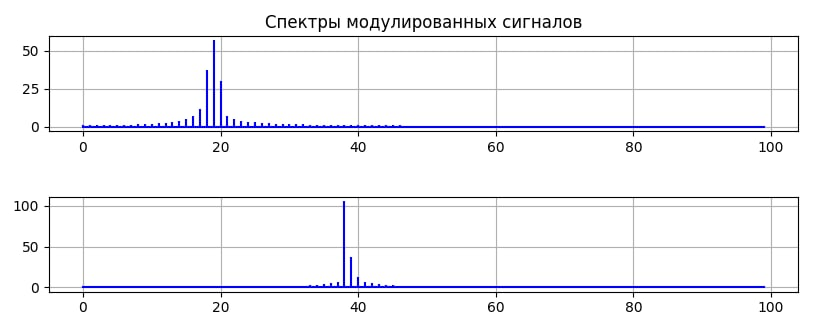
t — время;

ω — несущая частота.

Модулированные сигналы изображены на рис. 4.8.

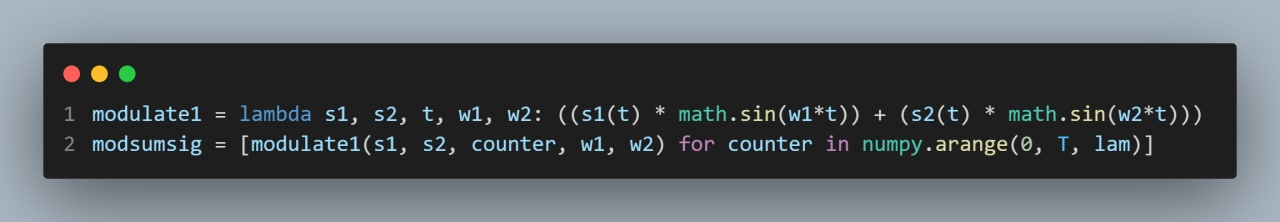


Для наглядности дальнейших действий построим спектры получившихся сигналов. Они изображены на рис. 4.9.



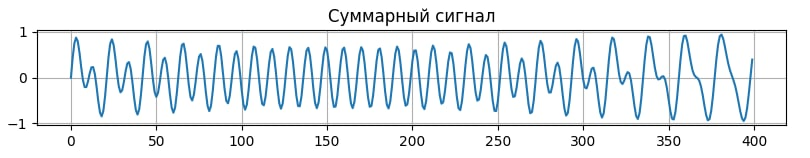
Как можно заметить спектр модулированного сигнала сильно ограничен

по ширине, за счёт чего и происходит частотное объединение сигналов. По сути этот процесс представляет линейное сложение 2х сигналов во временной области (и как мы позже увидим, это верно и для частотной области). Объединение сигналов будем проводить с помощью функции



где s — массив из функций входных сигналов.

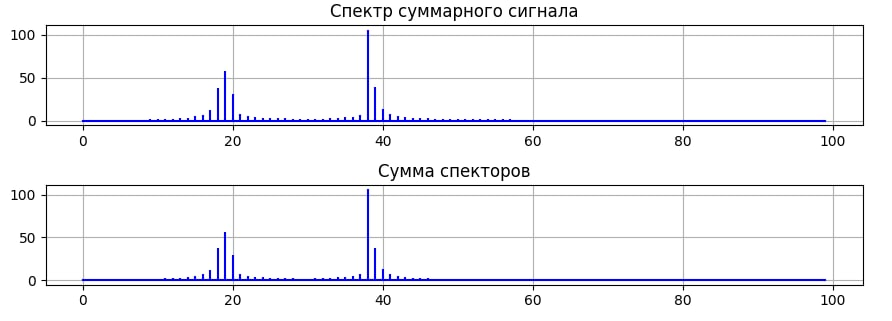
Получившийся сигнал, готовый к передаче изображён на рис. 4.10



На рис. 4.11 сверху изображён спектр, посчитанный для суммарного

сигнала, снизу изображена линейная сумма спектров исходных сигналов. Как

можно заметить между ними нет никаких различий.



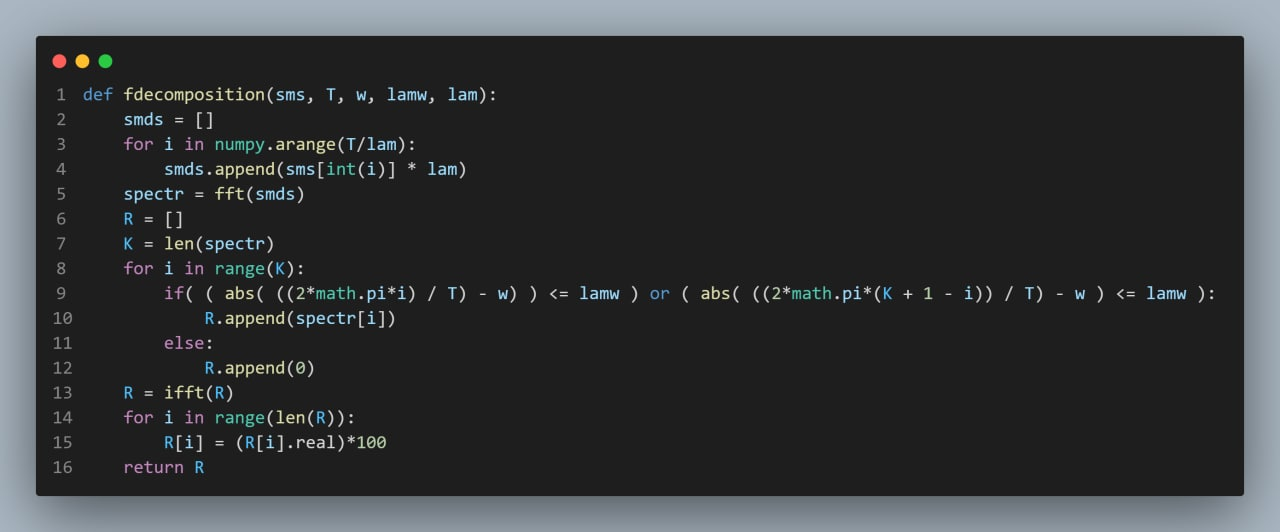
Разделение будем осуществлять с помощью функции (4.4). Она

основывается на преобразовании Фурье и состоит из нескольких шагов:

дискретизовать сигнал, посчитать его спектр, отфильтровать его часть, и с

помощью обратного преобразования Фурье вернуться к сигналу во временной

области. В реальности это решается намного проще, с помощью полосновыделяющего фильтра.



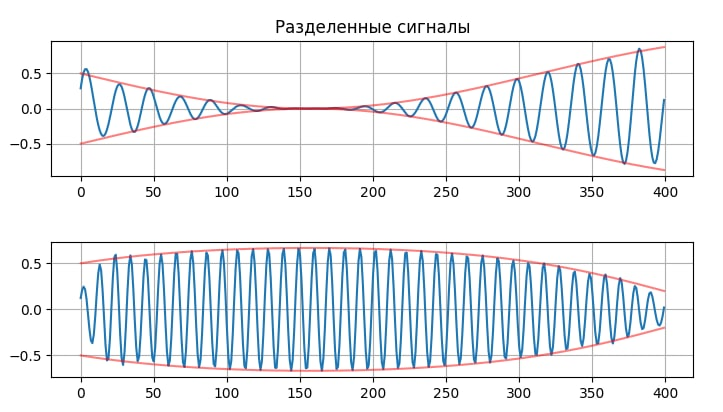
где sms — принятый сигнал.

Демодулированные сигналы изображены на рис. 4.12. Для наглядности

приведены огибающие, соответствующие исходному сигналу. Как можно

заметить существует небольшие отклонения по краям сигналов, что

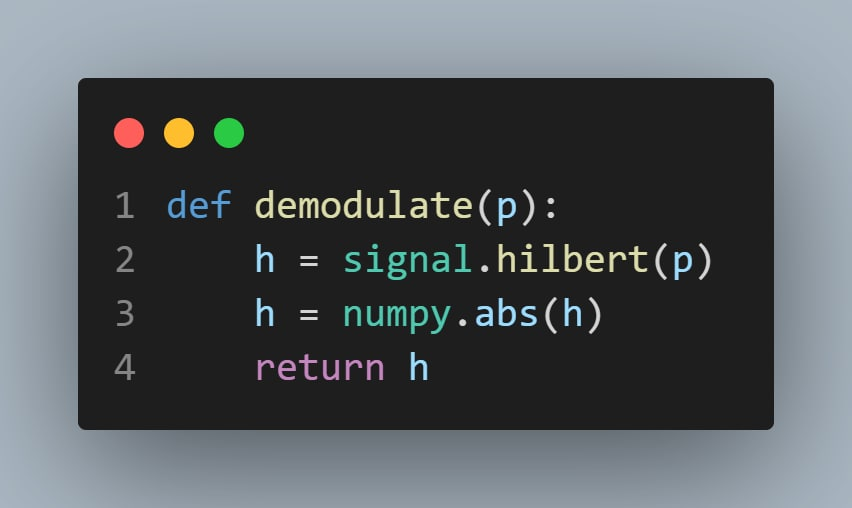
обуславливается погрешностью дискретных преобразований Фурье.



Демодуляция сигнала подразумевает собой построение огибающей

сигнала, которое основывается на добавлении ортогонального дополнения

сигнала, полученного с помощью преобразования Гилберта



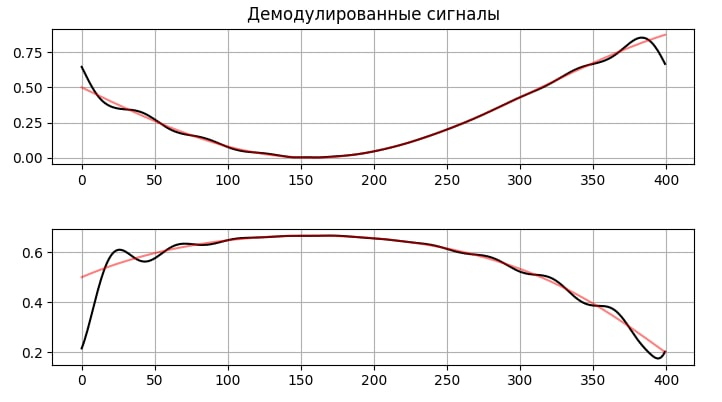
Демодулированные сигналы изображены на рис. 4.13. Для наглядности

исходный сигнал изображён на том же графике, менее насыщенным цветом. Как

можно заметить принятый сигнал по краям отличается от исходного. В

основном это объясняется погрешностями преобразования Фурье, и в реальной

жизни для бесконечного сигнала не характерно.

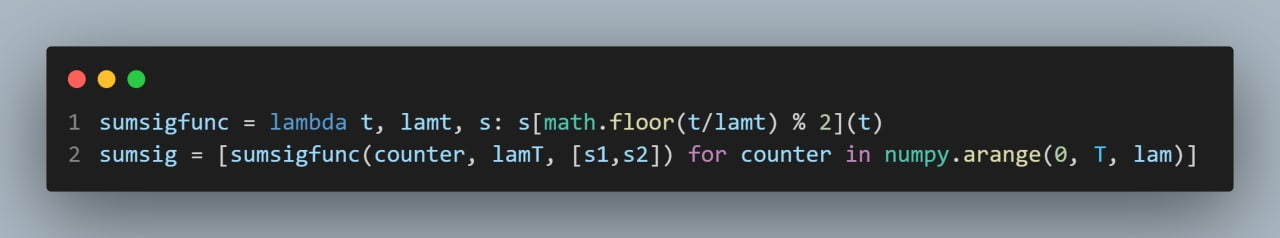


# **4 Принцип временного объединения и разделения каналов на практике**

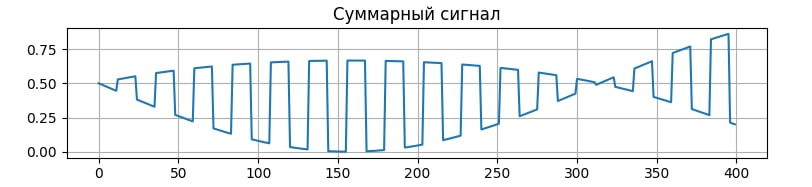
Для временного объединения каналов не требуется никакой

предварительной подготовки. Объединение будет происходить с помощью

функции (4.6).



Полученный сигнал изображён на рис. 4.14.

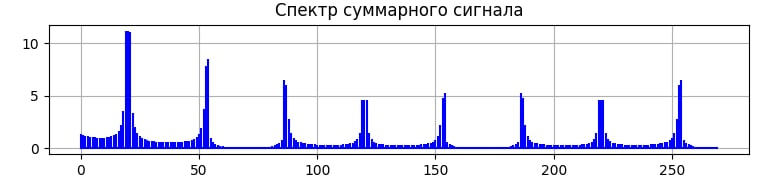


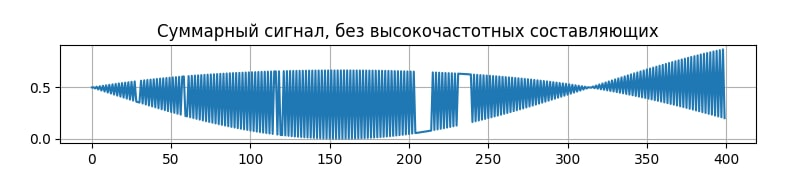
Спектр полученного сигнала изображён на рис. 4.15. Как можно

заметить полученный сигнал обладает достаточно широким спектром. При

использовании канала передачи с ограниченной способностью передачи

высокочастотных сигналов, то сигнал исказится (рис. 4.16).

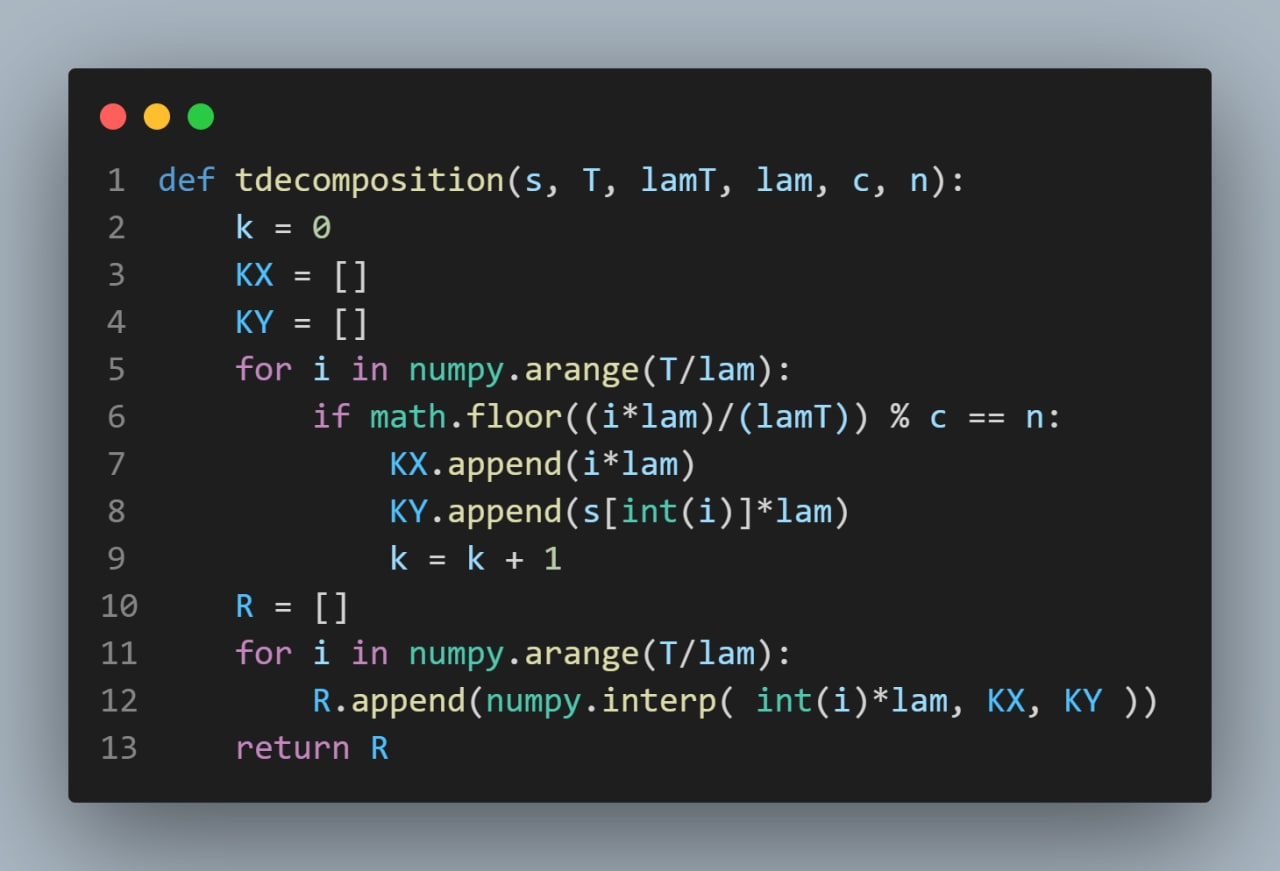




Восстановим исходные сигналы с помощью функции (4.7).

Недостающие участки сигналов восстановим с помощью линейной

интерполяции.

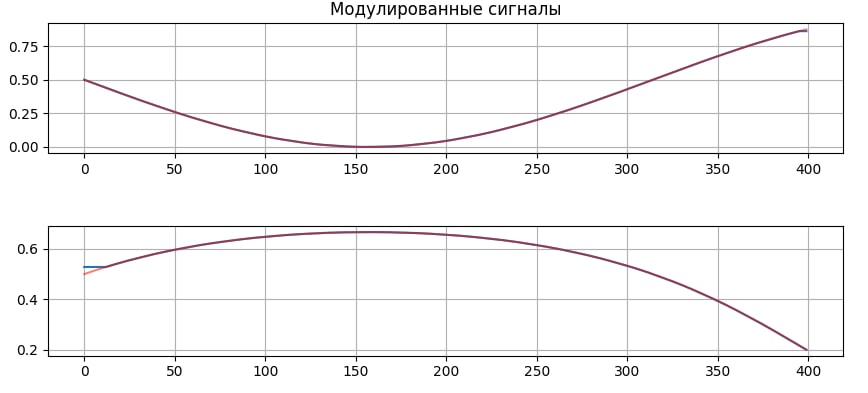


где с — количество объединённых сигналов;

n — номер извлекаемого сигнала.

Разделённые сигналы изображены на рис. 4.17. Как можно убедиться

почти идеально совпадают с исходными сигналами



Так же стоит смоделировать сигналы, полученные разделением

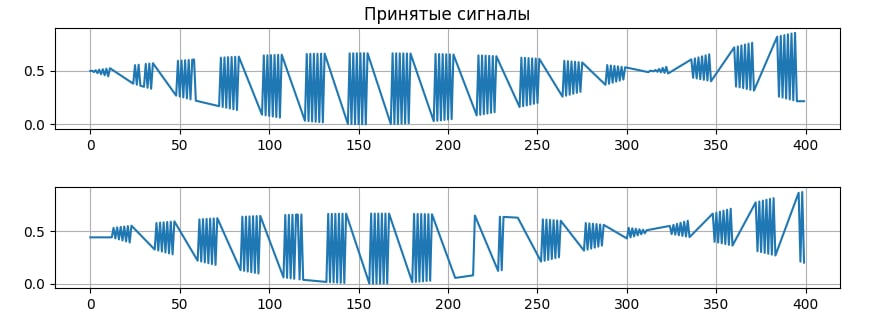
искажённого сигнала. Они изображены на рис. 4.18. Как можно убедиться

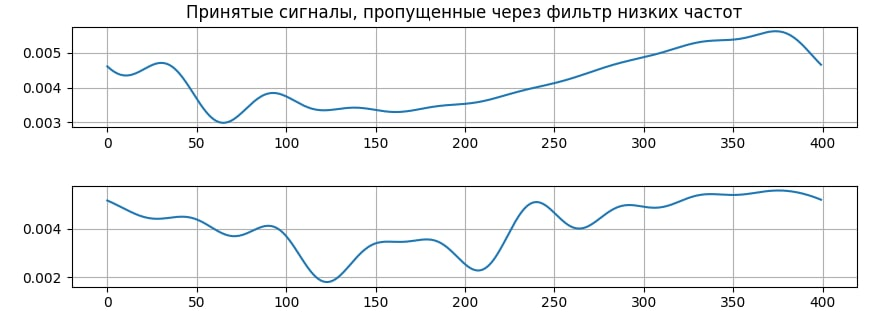
полученные сигналы сильно искажены. Добиться более точного варианта

68

можно воспользовавшись полосно-выделяющим фильтром низких частот

(рис. 4.19), но даже с ним сигнал отличается от исходного.





# **7 Заключение**

# **Приложение А**

Листинг программы

Файл CodeLab.py

import math

from numbers import Real

import numpy

from scipy.fft import fft

import scipy

from scipy.fft import ifft

import scipy.signal as signal

def fcomposition(t, s):

R = 0

for i in range(len(s)):

F = s[i]

R = R + F(t)

return R

def demodulate(p, lam, T):

h = signal.hilbert(p)

h = numpy.abs(h)

return h

def fdecomposition(sms, T, w, lamw, lam):

smds = []

for i in numpy.arange(T/lam):

smds.append(sms[int(i)] \* lam)

spectr = fft(smds)

R = []

K = len(spectr)

for i in range(K):

if( ( abs( ((2\*math.pi\*i) / T) - w) ) <= lamw ) or ( abs( ((2\*math.pi\*(K + 1 - i)) / T) - w ) <= lamw ):

R.append(spectr[i])

else:

R.append(0)

R = ifft(R)

for i in range(len(R)):

R[i] = (R[i].real)\*100

return R

def filtr(sms, T, w, lamw, lam):

smds = []

for i in numpy.arange(T/lam):

smds.append(sms[int(i)] \* lam)

spectr = fft(smds)

R = []

K = len(spectr)

for i in range(K):

if( ( abs( ((2\*math.pi\*i) / T) - w) ) <= lamw ) or ( abs( ((2\*math.pi\*(K + 1 - i)) / T) - w ) <= lamw ):

R.append(spectr[i])

else:

R.append(0)

R = ifft(R)

for i in range(len(R)):

R[i] = (R[i].real)

return R

def tdecomposition(s, T, lamT, lam, c, n):

k = 0

KX = []

KY = []

for i in numpy.arange(T/lam):

if math.floor((i\*lam)/(lamT)) % c == n:

KX.append(i\*lam)

KY.append(s[int(i)]\*lam)

k = k + 1

R = []

for i in numpy.arange(T/lam):

R.append(numpy.interp( int(i)\*lam, KX, KY ) \* 100)

return R

Файл lab4.py

import math

from signal import signal

import numpy

import codelab

import matplotlib.pyplot as plt

from scipy.fft import fft

modulate = lambda sd, t, w: sd(t)\*math.sin(w\*t)

# Наш вариант

s1 = lambda t: (1 - math.sin(t)) / 2

s2 = lambda t: (1 + math.sin(t)) / (2 + math.sin(t))

lam = 0.01

lamT = 0.12

T = 4

w1 = 30

lamw = 15

w2 = w1+2\*lamw

# Вариант из методички.

# s1 = lambda t: 1 + math.sin(t \* math.sin(t))

# s2 = lambda t: 1 + math.cos(t \* math.sin(t))

# lam = 0.01

# lamT = 0.1

# T = 6

# w1 = 20

# lamw = 15

# w2 = w1+2\*lamw

sig1 = [s1(counter) for counter in numpy.arange(0, T, lam)]

sig2 = [s2(counter) for counter in numpy.arange(0, T, lam)]

plt.figure()

plt.subplot(411)

plt.plot(sig1)

plt.title('Исходные сигналы')

plt.grid(True)

plt.subplot(412)

plt.plot(sig2)

plt.grid(True)

modsig1 = [modulate(s1,counter, w1) for counter in numpy.arange(0, T, lam)]

modsig2 = [modulate(s2,counter, w2) for counter in numpy.arange(0, T, lam)]

plt.subplot(413)

plt.plot(modsig1)

plt.plot([ -i for i in sig1 ], alpha=0.5, color='r')

plt.plot(sig1, alpha=0.5, color='r')

plt.title('Модулированные сигналы')

plt.grid(True)

plt.subplot(414)

plt.plot(modsig2)

plt.plot([ -i for i in sig2 ], alpha=0.5, color='r')

plt.plot(sig2, alpha=0.5, color='r')

plt.grid(True)

plt.figure()

plt.subplot(511)

plt.stem([abs(i) for i in fft(modsig1)][:100], linefmt='blue', markerfmt='none', basefmt='blue' )

plt.title('Спектры модулированных сигналов')

plt.grid(True)

plt.subplot(512)

plt.stem([abs(i) for i in fft(modsig2)][:100], linefmt='blue', markerfmt='none', basefmt='blue' )

plt.grid(True)

#modulate1 = lambda s1, s2, t, w1, w2, w3, w4: ((s1(t))\*math.sin(w1\*t) + (s2(t)\*math.sin(w2\*t)))

modulate1 = lambda s1, s2, t, w1, w2: ((s1(t) \* math.sin(w1\*t)) + (s2(t) \* math.sin(w2\*t)))

modsumsig = [modulate1(s1, s2, counter, w1, w2) for counter in numpy.arange(0, T, lam)]

plt.subplot(513)

plt.plot(modsumsig)

plt.title('Суммарный сигнал')

plt.grid(True)

plt.subplot(514)

plt.stem([abs(i) for i in fft(modsumsig)][:100], linefmt='blue', markerfmt='none', basefmt='blue' )

plt.title('Спектр суммарного сигнала')

plt.grid(True)

plt.subplot(515)

plt.stem([abs(i) for i in fft(modsig1)][:30] + [abs(i) for i in fft(modsig2)][30:100], linefmt='blue', markerfmt='none', basefmt='blue')

plt.title('Сумма спекторов')

plt.grid(True)

plt.figure()

decompositionsig1 = codelab.fdecomposition(modsumsig, T, w1, lamw, lam)

decompositionsig2 = codelab.fdecomposition(modsumsig, T, w2, lamw, lam)

plt.subplot(411)

plt.plot(decompositionsig1)

plt.plot([ -i for i in sig1 ], alpha=0.5, color='r')

plt.plot(sig1, alpha=0.5, color='r')

plt.title('Разделенные сигналы')

plt.grid(True)

plt.subplot(412)

plt.plot(decompositionsig2)

plt.plot([ -i for i in sig2 ], alpha=0.5, color='r')

plt.plot(sig2, alpha=0.5, color='r')

plt.grid(True)

plt.subplot(413)

plt.plot(codelab.demodulate(decompositionsig1.real, lam, T), color='black')

#plt.plot([ -i for i in sig1 ], alpha=0.5, color='r')

plt.plot(sig1, alpha=0.5, color='r')

plt.title('Демодулированные сигналы')

plt.grid(True)

plt.subplot(414)

plt.plot(codelab.demodulate(decompositionsig2.real, lam, T), color='black')

#plt.plot([ -i for i in sig2 ], alpha=0.5, color='r')

plt.plot(sig2, alpha=0.5, color='r')

plt.grid(True)

plt.figure()

sumsigfunc = lambda t, lamt, s: s[math.floor(t/lamt) % 2](t)

sumsig = [sumsigfunc(counter, lamT, [s1,s2]) for counter in numpy.arange(0, T, lam)]

plt.subplot(411)

plt.plot(sumsig)

plt.title('Суммарный сигнал')

plt.grid(True)

plt.subplot(412)

plt.stem([abs(i) for i in fft(sumsig)][30:300], linefmt='blue', markerfmt='none', basefmt='blue' )

plt.title('Спектр суммарного сигнала')

plt.grid(True)

plt.subplot(413)

plt.plot(codelab.tdecomposition(sumsig,T,lamT,lam,2,0))

plt.plot(sig1, alpha=0.5, color='r')

plt.title('Модулированные сигналы')

plt.grid(True)

plt.subplot(414)

plt.plot(codelab.tdecomposition(sumsig,T,lamT,lam,2,1))

plt.plot(sig2, alpha=0.5, color='r')

plt.grid(True)

plt.figure()

sumsigfunc = lambda t, lamt, s: s[math.floor(t/lamt) % 2](t)

sumsig = [sumsigfunc(counter, lam, [s1,s2]) for counter in numpy.arange(0, T, lam)]

plt.subplot(511)

plt.plot(sumsig)

plt.title('Суммарный сигнал, без высокочастотных составляющих')

plt.grid(True)

plt.subplot(512)

plt.plot(codelab.tdecomposition(sumsig,T,lamT,lam,2,0))

plt.title('Принятые сигналы')

plt.grid(True)

plt.subplot(513)

plt.plot(codelab.tdecomposition(sumsig,T,lamT,lam,2,1))

plt.grid(True)

signal1 = codelab.tdecomposition(sumsig,T,lamT,lam,2,0)

signal2 = codelab.tdecomposition(sumsig,T,lamT,lam,2,1)

decompositionsig1 = codelab.filtr(signal1, T, 0.00000000001, lamw, lam)

decompositionsig2 = codelab.filtr(signal2, T, 0.00000000001, lamw, lam)

plt.subplot(514)

plt.plot(decompositionsig1)

#plt.plot(sig1, alpha=0.5, color='r')

plt.title('Принятые сигналы, пропущенные через фильтр низких частот')

plt.grid(True)

plt.subplot(515)

plt.plot(decompositionsig2)

#plt.plot(sig2, alpha=0.5, color='r')

plt.grid(True)

plt.show()