Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №2 по дисциплине

"Физические основы информационных технологий"

Название: Искажение сигнала в волноводе

|  |  |
| --- | --- |
| Фамилия И.О.: | Павлов Д.Р. |
| группа: | 9303 |
| Преподаватель: | Альтмарк А.М. |
| Итоговый балл: |  |
|  |  |

Крайний срок сдачи: 9.11.21

.

Санкт-Петербург 2021

Условие ИДЗ

Радиоимпульсы (рис.1), предназначенные для последовательной передачи двоичного кода, распространяются через цилиндрический диэлектрический волновод (рис.2). Оценить дистанцию прохождения последовательности радиоимпульсов в цилиндрическом диэлектрическом волноводе, определяемую искажением сигнала из-за волноводной дисперсии. При расчетах учитывать только первую моду. Исходными данными являются параметры волновода (*R*- радиус волновода, -диэлектрическая проницаемость), параметры последовательности радиоимпульсов (,,) и порог, разделяющий уровень "0" и "1". Порог определяется следующей формулой:

,

где *S* - уровень сигнала между импульсами,  - максимальное значение сигнала внутри импульса





Рисунок.1



Рисунок.2

Этапы решения задания

1. Определение фазовой скорости от частоты путем интерполирования или аппроксимации данных. Определение пороговой частоты (частоты, при которой фазовая скорость уходит в бесконечность)
2. Построение спектра сигнала. Спектр должен находится практически во всем диапазоне графика фазовой скорости
3. Обратное преобразование Фурье с учетом волноводной дисперсии.
4. Построение зависимости последовательности импульсов от расстояния. Анализ областей, расположенных между сигналами, на превышение порога. Остановка программы с выводом дистанции в качестве ответа

В папке LR2 должен находиться Word-файл с отчетом, а также файл с кодом (Python, Mathcad, Mathematica). Для лучшего понимания отчетности смотрите папку “Пример организации яндекс-папки студентов”.

**10 вариант**.

**Выполнение работы**

Решаем дисперсионное уравнение для поиска функции зависимости фазовой скорости от частоты. Получившийся массив интерполируем и получаем следующую функцию, представленную на рис.3.

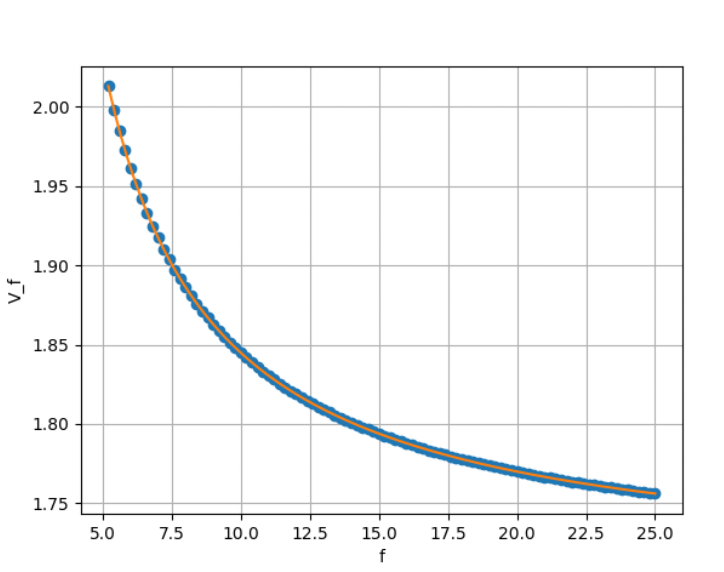


Рисунок 3 – Дисперсионная кривая для цилиндрического волновода

Далее разложим заданный сигнал, представленный на рис.4, на спектр.

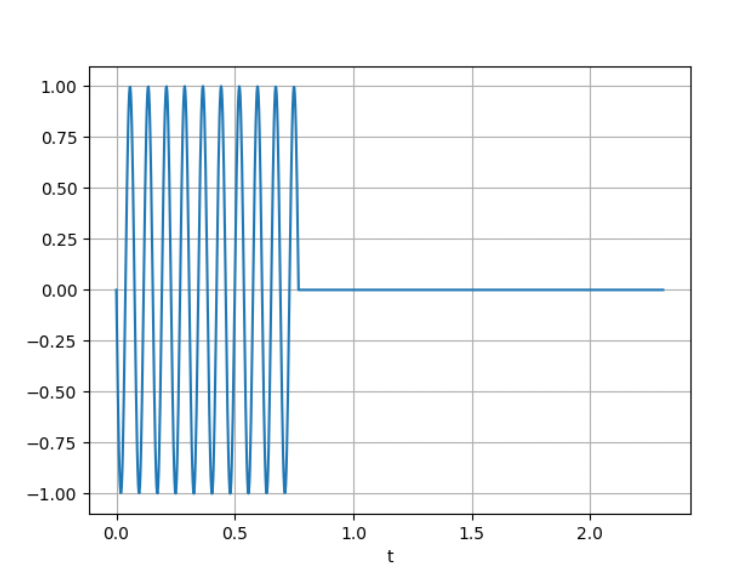


Рисунок 4 – Заданный сигнал

По заданной формуле считаем спектр и получаем следующий график, представленный на рис.5.

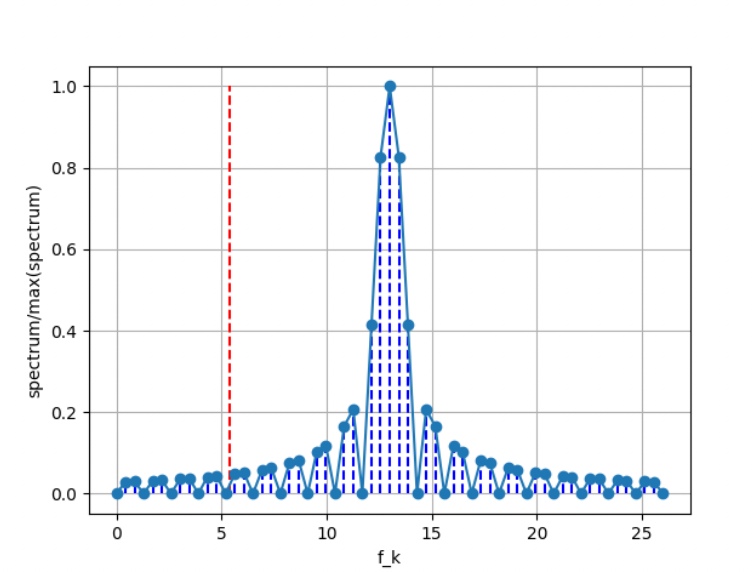


Рисунок 5 – График спектра

Воспользуемся обратным преобразованием Фурье и восстановим сигнал. Зависимость сигнала от времени см. на рис. 6, зависимость сигнала от расстояния см. на рис. 7.

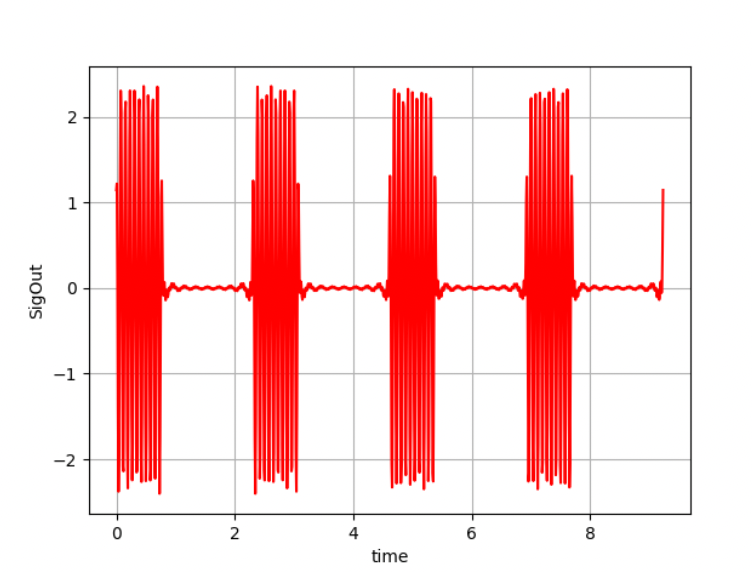
****

Рисунок 6 – Зависимость сигнала от времени.

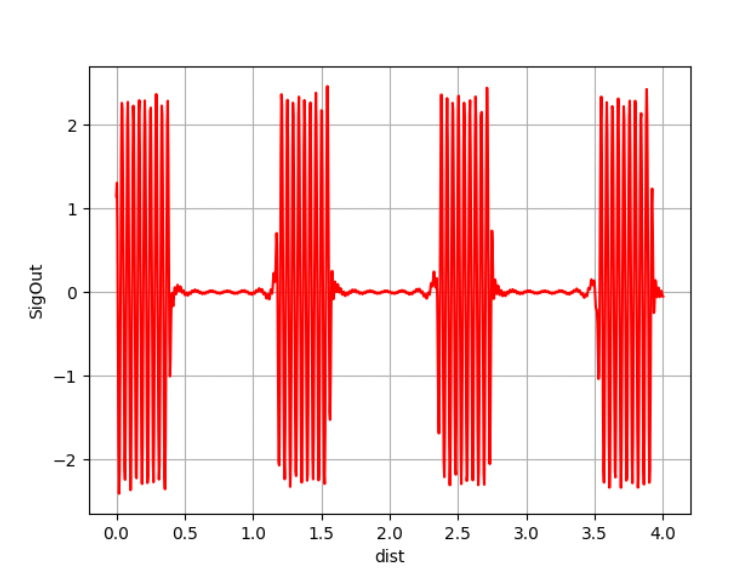
****

Рисунок 7 – Зависимость сигнала от расстояния.

На рис. 8 можно увидеть график зависимости сигнала от времени для z = 100, на котором превышения между областями нет.

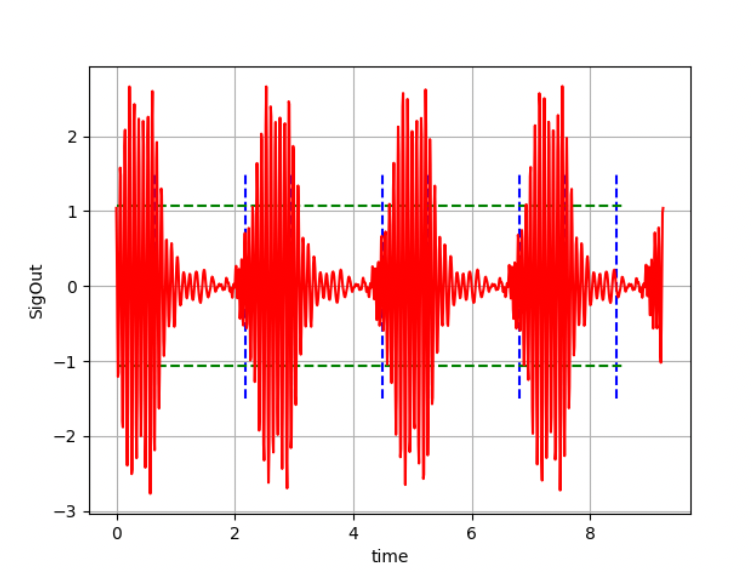


Рисунок 8 – График зависимости сигнала от времени для z=100.

**ПРОГРАММА MAIN.PY**

import matplotlib.pyplot as plt

import numpy as np

import scipy.special

import scipy.optimize

from scipy.interpolate import interp1d

from scipy.constants import speed\_of\_light

R = 1

e = 5.4

f0 = 13

t\_1 = 1 / f0 \* 10

t\_2 = 1 / f0 \* 30

t\_3 = 1 / f0

P = 0.4

c = speed\_of\_light

def Heaviside(x): # Ступенчатая функция

return 0 if x < 0 else 1

def signal(t): # Описание сигнала

return np.sin(f0 \* 2 \* np.pi \* t) \* (Heaviside(t - t\_1) - Heaviside(t))

def Signal(t, z, N1=60):

sum = 0

for k in range(0, N1 + 1):

if f\_gr < f\_k[k] < fl\_k[99]:

sum += spectrum2[k] \* np.cos(

f\_k[k] \* 2 \* np.pi \* t - f\_k[k] \* 2 \* np.pi / (v\_f(f\_k[k]) \* c / (10 \*\* 9)) \* z)

return sum

def Vfc\_plot(v\_f, v\_fk, xnew, fl\_k):

fig, ax = plt.subplots()

ax.plot(fl\_k, v\_fk[1:], 'o', xnew, v\_f(xnew), '-')

ax.set\_xlabel('f')

ax.set\_ylabel('V\_f')

ax.grid(True)

plt.show()

def impulse\_plot():

N = 2 \*\* 10

time = [t\_2 \* i / (N - 1) for i in range(0, N)] # Моменты времени для построения графика

point\_s = [signal(i) for i in time]

fig, ax = plt.subplots()

ax.plot(time, point\_s)

ax.set\_xlabel('t')

ax.grid(True)

plt.show()

def Inverse\_Fourier\_transform\_plot(time, sigout):

fig, ax = plt.subplots()

ax.plot(time, sigout, 'r')

ax.set\_xlabel('time')

ax.set\_ylabel('SigOut')

ax.grid(True)

plt.show()

"""1. Построение графика зависимости фазовой скорости от частоты. Опредление пороговой частоты"""

def first\_task():

def dispersion\_equation(v\_fc): # Дисперсионное уравнение

return scipy.special.j0(R \* frequencies[i] / c \* np.sqrt(e - 1 / v\_fc \*\* 2))

frequencies = [2 \* np.pi \* (5 + 20 \* i / 100) \* 10 \*\* 9 for i in range(101)] # Частоты для поиска фазовых скоростей

v\_fc = 2 # Начальное приближение

i = 0

res = scipy.optimize.root(dispersion\_equation, v\_fc) # Поиск первой фазовой скорости

v\_f0 = res.x[0]

v\_fk = [v\_f0] # Фазовые скорости

fl\_k = [] # Частоты f

# Поиск последующих фазовых скоростей

for k in range(1, 101):

i += 1

v\_fc = v\_fk[k - 1]

res = scipy.optimize.root(dispersion\_equation, v\_fc, method='lm')

v\_fk.append(res.x[0])

fl\_k.append(frequencies[i] / (2 \* np.pi \* 10 \*\* 9))

f\_gr = fl\_k[1] # Граничная частота

""" Апроксимация массива фазовых скоростей """

v\_f = interp1d(fl\_k, v\_fk[1:]) # Функция фазовой скорости от частоты

xnew = np.linspace(fl\_k[0], fl\_k[-1], num=500, endpoint=True)

Vfc\_plot(v\_f, v\_fk, xnew, fl\_k)

""" Построение графика импульса """

impulse\_plot()

return f\_gr, fl\_k, v\_f

"""2. Построение спектра сигнала """

def second\_task(f\_gr):

def sin\_x(x):

return 1 if x == 0 else np.sin(x) / x

def spectrum(f): # Определение функции спектра радиоимпульса

return sin\_x((f - f0) \* 2 \* np.pi \* t\_1 / 2) \* t\_1

N1 = 60

f\_k = [k / t\_2 for k in range(0, N1 + 1)] # Частоты, задающие интервалы между импульсами

spectrum1 = [np.abs(spectrum(f)) for f in f\_k]

spectrum2 = [spectrum(f) for f in f\_k]

point\_spec = [s / max(spectrum1) for s in spectrum1]

# График спектра

fig, ax = plt.subplots()

ax.plot(f\_k, point\_spec, 'o-')

ax.vlines(f\_gr, 0, 1, colors='r', linestyles='dashed', label="f\_gr")

for k in range(0, len(f\_k)):

ax.vlines(f\_k[k], 0, point\_spec[k], colors="b", linestyles='dashed')

ax.set\_xlabel('f\_k')

ax.set\_ylabel('spectrum/max(spectrum)')

ax.grid(True)

plt.show()

return f\_k, spectrum2

""" 3. Обратное преобразование Фурье"""

def third\_task():

N = 1000

z = 0

time = [4 \* t\_2 \* i / (N - 1) for i in range(0, N)]

sigout = [Signal(t - t\_1 / 2, z) for t in time]

Inverse\_Fourier\_transform\_plot(time, sigout)

"""4. Построение зависимости последовательности импульсов от расстояния."""

def fourth\_task():

N = 1000

Number = 0

dist = [Number + i / N\*4 for i in range(0, N + 1)]

time = [4 \* t\_2 \* i / (N - 1) for i in range(0, N)]

time1 = 0

sigout2 = [Signal(time1 + t\_1 / 2, d) for d in dist]

fig, ax = plt.subplots()

ax.plot(dist, sigout2, 'r')

ax.set\_xlabel('dist')

ax.set\_ylabel('SigOut')

ax.grid(True)

plt.show()

# Анализ областей между сигналами на превышении порога

time\_between\_pulse = t\_2 - t\_1

eps = 0.0025

flag = 0

for z in range(100, 110, 1):

if flag:

break

fig, ax = plt.subplots()

current\_time = z \* 0.0845

sigout = [Signal(t - t\_1 / 2, z) for t in time]

lines = []

for i in range(0, 8):

lines.append(current\_time % 8.5712)

ax.vlines(current\_time % 8.5712, -1.5, 1.5, colors='b', linestyles='dashed')

current\_time += time\_between\_pulse if i % 2 else t\_1

ax.hlines(P \* max(sigout), 0, 8.5712, colors="g", linestyles='dashed')

ax.hlines(- (P \* max(sigout)), 0, 8.5712, colors="g", linestyles='dashed')

lines.sort()

count = 1 if lines[0] > t\_1 or lines[0] == 0 else 0

for k in range(len(time)):

if lines[count] < time[k] < lines[count + 1]:

if np.abs(sigout[k]) >= P \* max(sigout) \

and time[k] - lines[count] > eps \

and lines[count + 1] - time[k] > eps:

print("distance =", z)

flag = 1

break

if time[k] > lines[count + 1]:

if count >= 5:

break

count += 2

ax.plot(time, sigout, 'r')

ax.set\_xlabel('time')

ax.set\_ylabel('SigOut')

ax.grid(True)

plt.show()

f\_gr, fl\_k, v\_f = first\_task()

f\_k, spectrum2 = second\_task(f\_gr)

third\_task()

fourth\_task()