

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 01

дисциплина: Сетевые технологии (09.03.03)

Студент: Стелина Петрити

Группа: НПИбд-02-21

МОСКВА

2023 г.

Цели работы

Изучение методов кодирования и модуляции сигналов с помощью высокоуровнного языка программирования Octave. Определение спектра и параметров сигнала. Демонстрация принципов модуляции сигнала на примере аналоговой амплитудной модуляции. Исследование свойства самосинхронизации сигнала.

1.3.1. Построение графиков в Octave

1.3.1.1. Постановка задачи

- Построить график функции $y = \sin x + \frac{1}{3} \sin 3x + \frac{1}{5} \sin 5x$ на интервале $[-10; 10]$, используя Octave и функцию plot. График экспортовать в файлы формата .eps, .png.

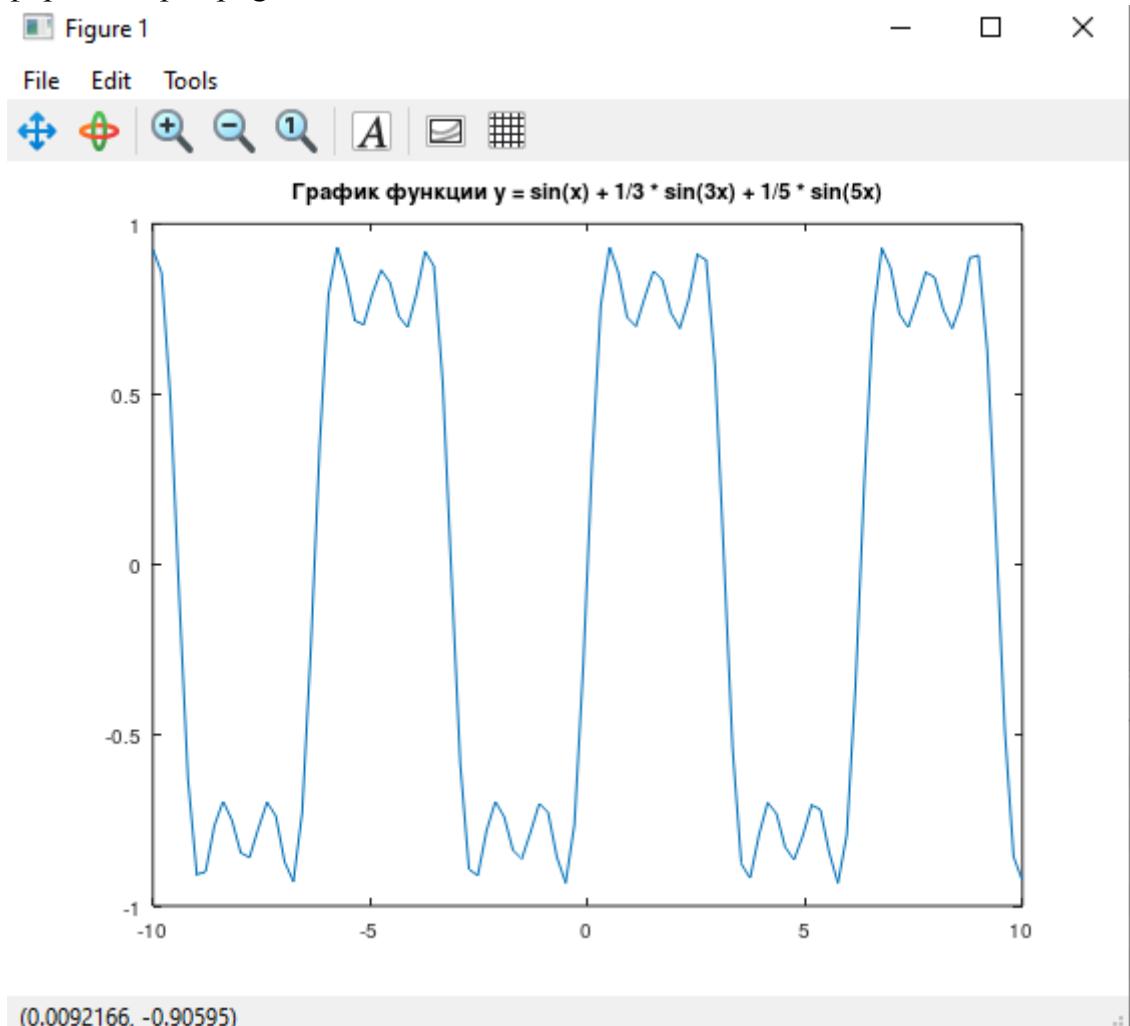


рис.1.3.1.1.1 График функции $y = \sin x + \frac{1}{3} \sin 3x + \frac{1}{5} \sin 5x$

```
Command Window
>> x = linspace(-10, 10);
>> y = sin(x) + 1/3 * sin(3*x) + 1/5 * sin(5*x);
>> plot(x, y);
>> plot(x, y);
>> title('График функции y = sin(x) + 1/3 * sin(3x) + 1/5 * sin(5x)');
>> |
```

код построен на основе требований. Для построения графика мы будем использовать `plot`.

рис.1.3.1.1.2 код в octave

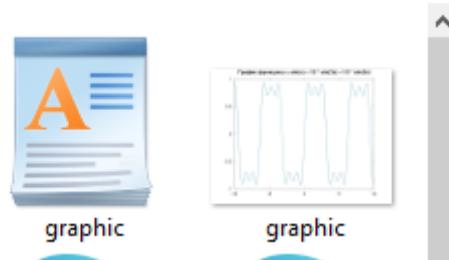


рис.1.3.1.1.3 График в файлы формата .eps, .png.

Чтобы сохранить его в форматах `eps` и `png`, мы можем щелкнуть файл в графических окнах и нажать `save as`.

2. Добавить график функции $y = \cos x + \frac{1}{3} \cos 3x + \frac{1}{5} \cos 5x$ на интервале $[-10; 10]$. График экспортовать в файлы формата `.eps`, `.png`.

Figure 1

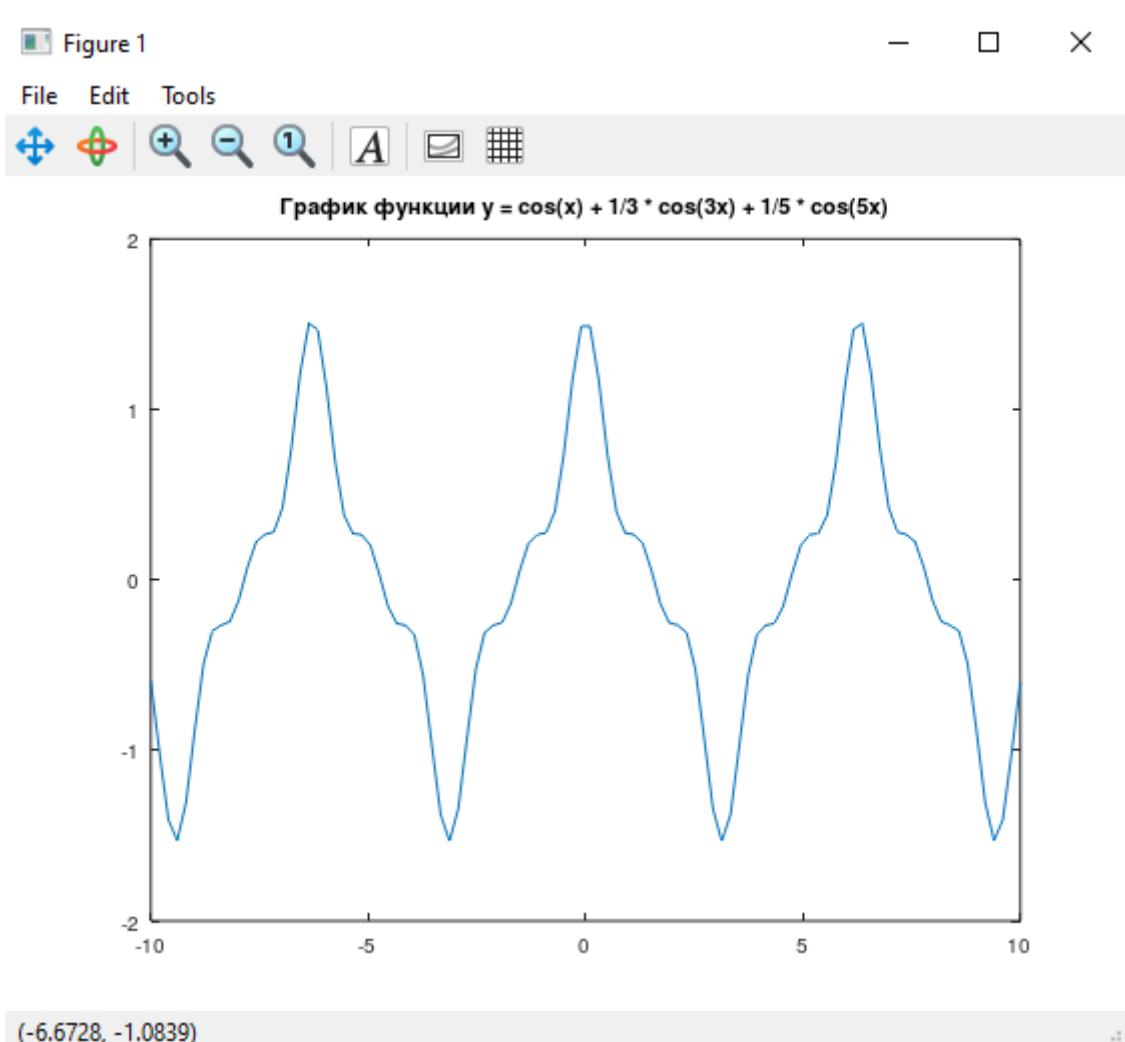


рис.1.3.1.1.4 График функции $y = \cos x + \frac{1}{3} \cos 3x + \frac{1}{5} \cos 5x$

Command Window

```
>> x = linspace(-10, 10);
>> y = cos(x) + 1/3 * cos(3*x) + 1/5 * cos(5*x);
>> plot(x, y);
>> title('График функции y = cos(x) + 1/3 * cos(3x) + 1/5 * cos(5x)');
>>
```

рис.1.3.1.1.5 код в octave

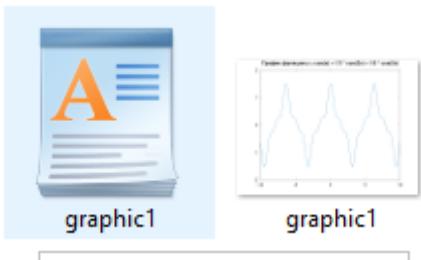


рис.1.3.1.1.6 График в файлы формата .eps, .png.

То же действие, что и в первом упражнении.

1.3.1.2. Выполнение работы

1. Запустите в вашей ОС Octave с оконным интерфейсом.
2. Перейдите в окно редактора. Воспользовавшись меню или комбинацией клавиш **ctrl + n** создайте новый сценарий. Сохраните его в ваш рабочий каталог с именем, например, `plot_sin.m`.

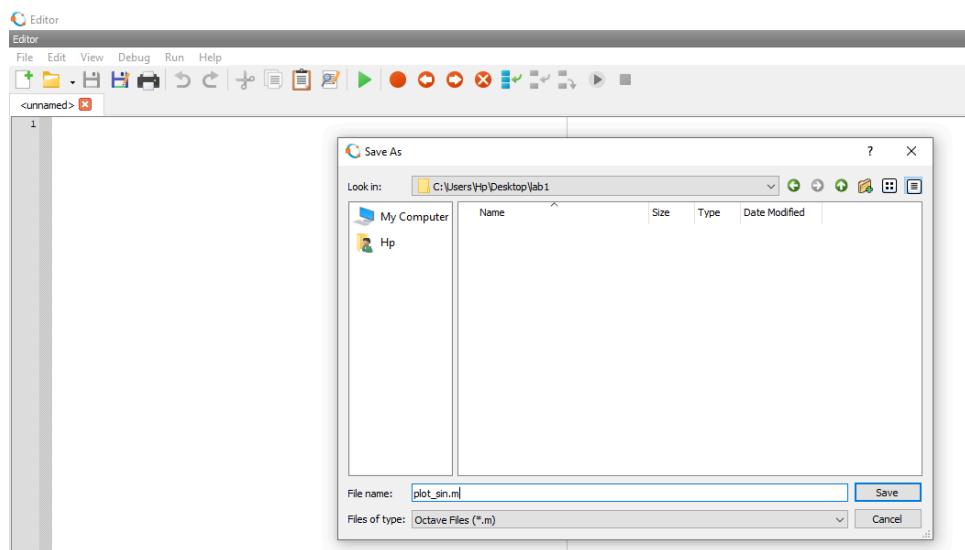


рис.1.3.1.2.1 окно редактора, создание `plot_sin.m`

Теперь мы будем работать в пространстве редактора. Это делается путем создания нового файла: например, имя файла `plot_sin`.

3. В окне редактора повторите следующий листинг по построению графика функции $y = \sin x + \frac{1}{3} \sin 3x + \frac{1}{5} \sin 5x$ на интервале $[-10; 10]$.

Мы создали, а затем ввели тот же код, что и в примере выше.

The screenshot shows a MATLAB code editor window titled 'plot_sin.m'. The code is a script for plotting a trigonometric function. It starts by defining a vector 'x' from -10 to 10 with a step of 0.1. Then it calculates 'y1' as the sum of three sine functions: sin(x), (1/3)*sin(3*x), and (1/5)*sin(5*x). The plot command is used to create a line graph with markers. The plot is labeled with 'x' and 'y'. A title is added to the plot. Finally, the plot is saved as both an EPS file and a PNG file.

```
1 x=-10:0.1:10;
2 y1=sin(x)+1/3*sin(3*x)+1/5*sin(5*x);
3 plot(x,y1, "-ok; y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);", "markersize", 4)
4 grid on;
5 xlabel('x');
6 ylabel('y');
7 title('y1=sin x+ (1/3)sin(3x)+(1/5)sin(5x)');
8 print ("plot-sin.eps", "-mono", "-FArial:16", "-deps")
9 print("plot-sin.png");
10
```

рис.1.3.1.2.2 код по построению графика

```
% Формирование массива x:  
x=-10:0.1:10;  
  
% Формирование массива y.  
y1=sin(x)+1/3*sin(3*x)+1/5*sin(5*x);  
  
% Построение графика функции:  
plot(x,y1, "-ok; y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);", "markersize", 4)  
  
% Отображение сетки на графике  
grid on;  
  
% Подпись оси X:  
xlabel('x');  
  
% Подпись оси Y:  
ylabel('y');  
  
% Название графика:  
title('y1=sin x+ (1/3)sin(3x)+(1/5)sin(5x)');  
  
% Экспорт рисунка в файл .eps:  
print ("plot-sin.eps", "-mono", "-FArial:16", "-deps")  
  
% Экспорт рисунка в файл .png:  
print("plot-sin.png");
```

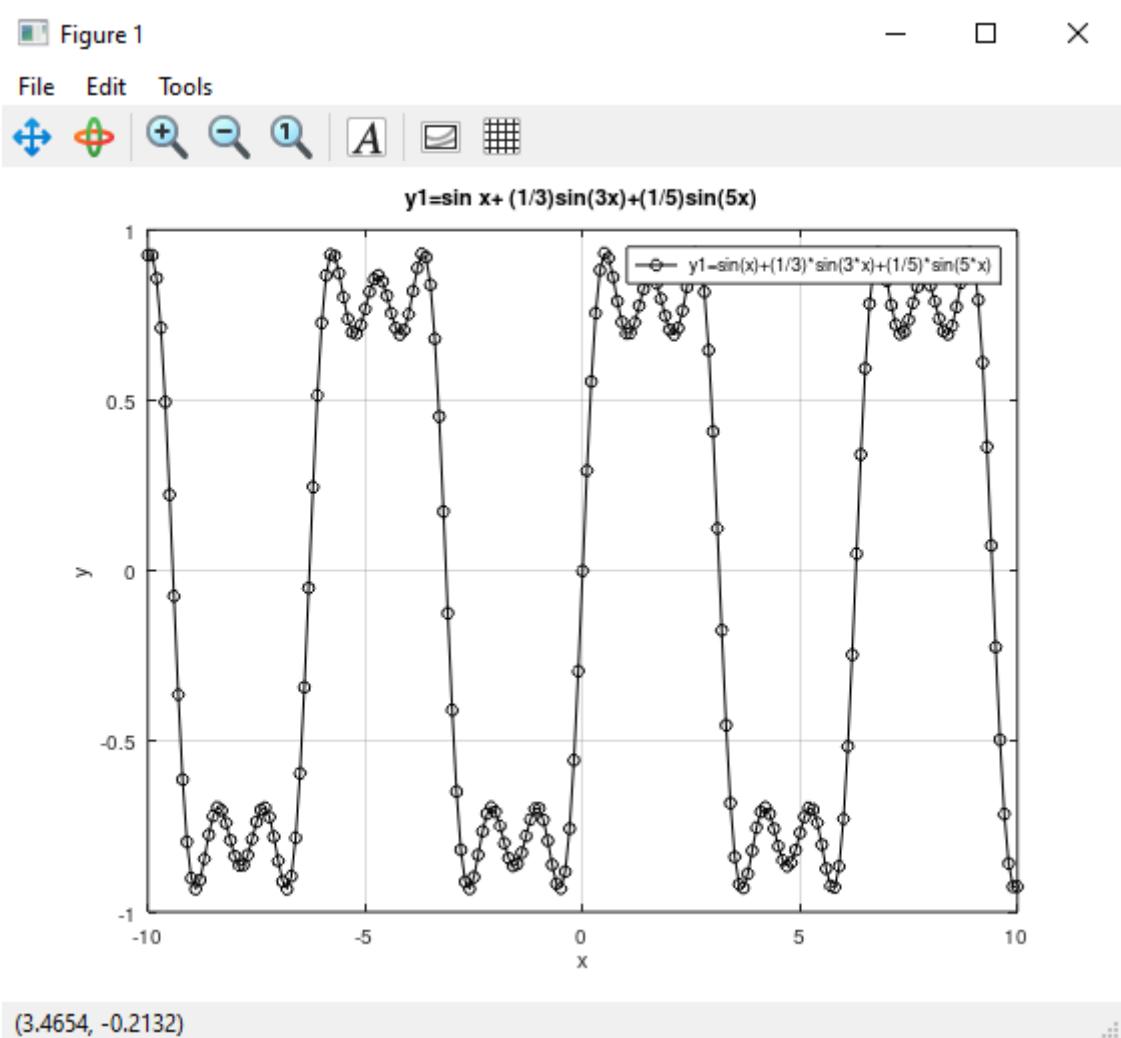


рис.1.3.1.2.3 График функции $y = \cos x + \frac{1}{3}\cos 3x + \frac{1}{5}\cos 5x$

4. Запустите сценарий на выполнение (воспользуйтесь соответствующим меню окна редактора или клавишей F5). В качестве результата выполнения кода должно открыться окно с построенным графиком и в вашем рабочем каталоге должны появиться файлы с графиками в форматах .eps, .png.

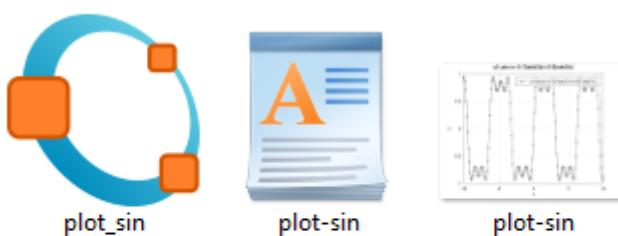
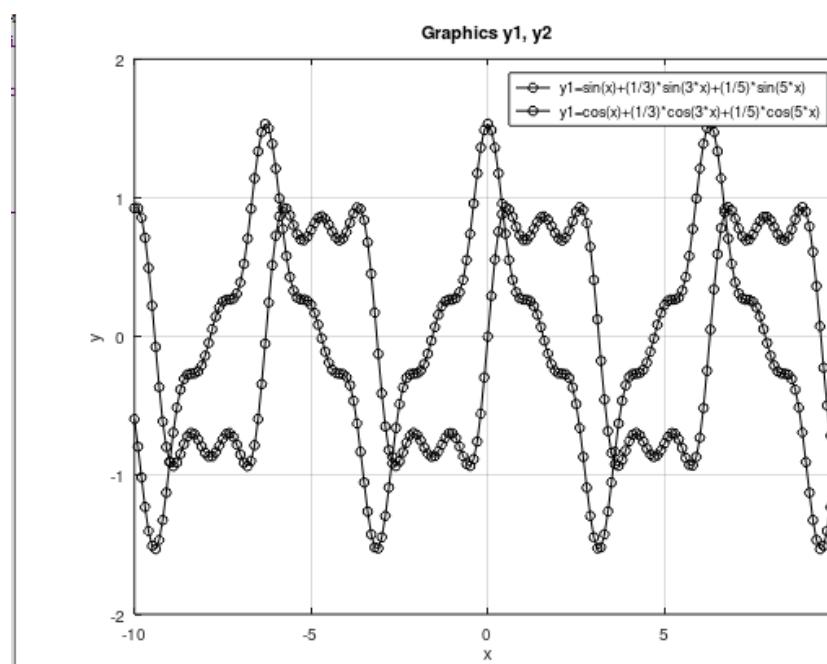


рис.1.3.1.2.4 файлы с графиками в форматах .eps, .png.

На этот раз, чтобы сохранить графику в формате png и eps, мы напишем две последние строки (рис..1.3.1.2.2)

5. Сохраните сценарий под другим названием и измените его так, чтобы на одном графике располагались отличающиеся по типу линий графики функций $y = \cos x + \frac{1}{3}\cos 3x + \frac{1}{5}\cos 5x$, $y = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x$.

```
%код
x=-10:0.1:10;
y1=sin(x)+1/3*sin(3*x)+1/5*sin(5*x);
y2=cos(x)+1/3*cos(3*x)+1/5*cos(5*x);
plot(x,y1,
      "y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);", "markersize",4)
hold on;
plot(x,y2,
      "y1=cos(x)+(1/3)*cos(3*x)+(1/5)*cos(5*x);", "markersize",4)
grid on;
xlabel('x');
ylabel('y');
title('Graphics y1, y2');
print ("plot-sin1.eps", "-mono", "-FArial:16", "-deps")
print("plot-sin1.png");
```



Чтобы добавить еще одно графическое изображение к другому, мы будем использовать команду `hold on`

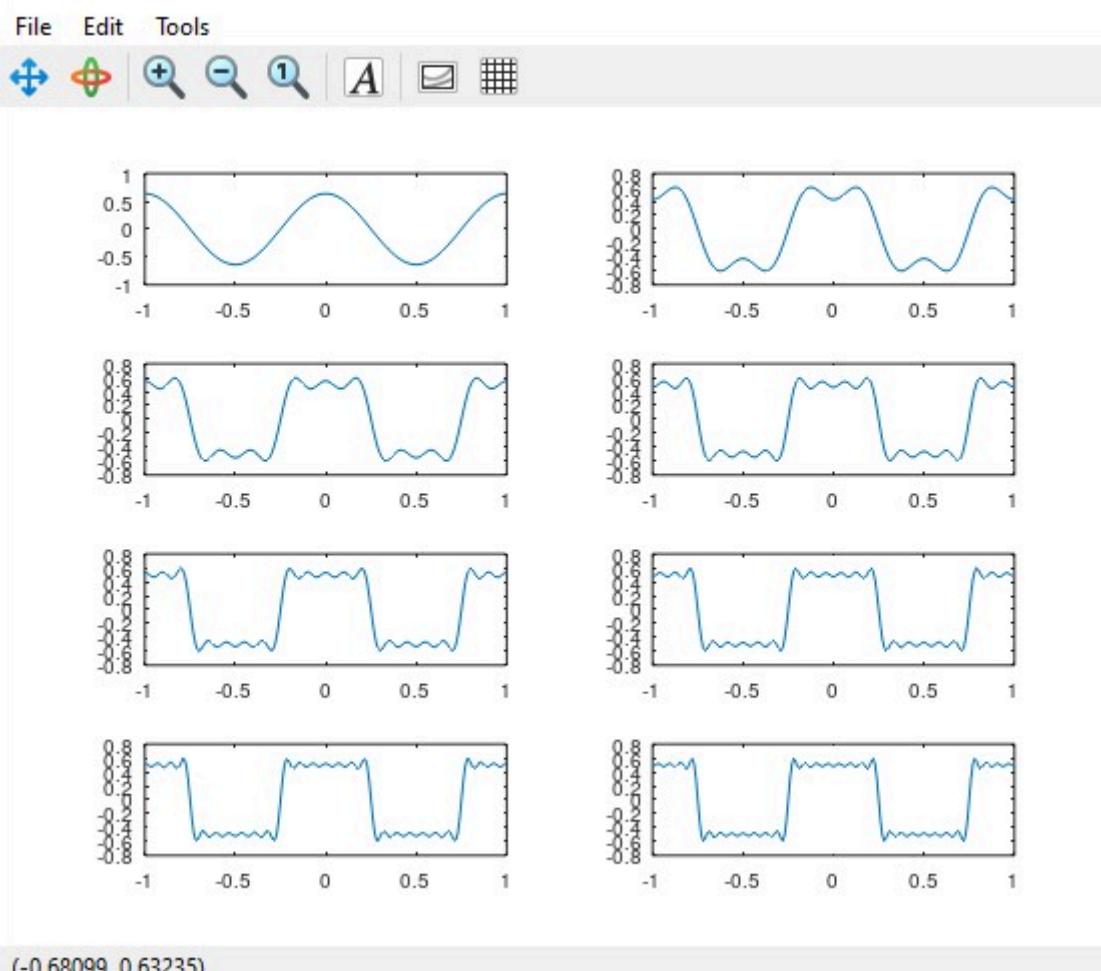
рис.1.3.1.2.5 график функций $y = \cos x + \frac{1}{3}\cos 3x + \frac{1}{5}\cos 5x$, $y = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x$.

1.3.2. Разложение импульсного сигнала в частичный ряд Фурье

1.3.2.1. Постановка задачи

1. Разработать код m-файла, результатом выполнения которого являются графики меандра реализованные с различным количеством гармоник.

Figure 1



1.3.2.1.1 Графики меандра, содержащего различное число гармоник

1.3.2.2.Выполнение работы

1. Создайте новый сценарий и сохраните его в ваш рабочий каталог с именем, например, meandr.m.



1.3.2.2.1 новый сценарий meandr.m.

2. В коде созданного сценария задайте начальные значения:

```
% meandr.m  
% количество отсчетов (гармоник):  
N=8;  
% частота дискретизации:  
t=-1:0.01:1;  
% значение амплитуды:
```

```
A=1;  
% период:
```

```
T=1;
```

3. Разложение импульсного сигнала в форме меандра в частичный ряд Фурье можно задать формулой

$$s(t) = A/2 + 2A/\pi (\cos(2\pi/T t) - 1/3 \cos(3 \cdot 2\pi/T t) + 1/5 \cos(5 \cdot 2\pi/T t) - \dots),$$

или формулой

$$s(t) = A/2 + 2A/\pi (\sin(2\pi/T t) + 1/3 \sin(3 \cdot 2\pi/T t) + 1/5 \sin(5 \cdot 2\pi/T t) + \dots).$$

т.е. в спектре присутствуют только нечётные гармоники.

Гармоники, образующие меандр, имеют амплитуду, обратно пропорциональную номеру соответствующей гармоники в спектре:

```
% амплитуда гармоник
```

```
nh=(1:N)*2-1;
```

% массив коэффициентов для ряда, заданного через cos:

```
Am=2/pi ./ nh;
```

```
Am(2:2:end) = -Am(2:2:end);
```

Далее задаём массив значений гармоник массив элементов ряда:

```
% массив гармоник:
```

```
harmonics=cos(2 * pi * nh' * t/T);
```

% массив элементов ряда:

```
s1=harmonics.*repmat(Am',1,length(t));
```

4. Далее для построения в одном окне отдельных графиков меандра с различным количеством гармоник реализуем суммирование ряда с накоплением и воспользуемся функциями subplot и plot для построения графиков:

```
% Суммирование ряда:
```

```
s2=cumsum(s1);
```

% Построение графиков:

```
for k=1:N
```

```
subplot(4,2,k)
```

```
plot(t, s2(k,:))
```

```
end
```

```
# код
```

```
N=8;
```

```
t=-1:0.01:1;
```

```
A=1;
```

```
T=1;
```

```
nh=(1:N)*2-1;
```

```
Am=2/pi ./ nh;
```

```
Am(2:2:end) = -Am(2:2:end);
```

```
harmonics=cos(2 * pi * nh' * t/T);
```

```
s1=harmonics.*repmat(Am',1,length(t));
```

```
s2=cumsum(s1);
```

```
for k=1:N
```

```
subplot(4,2,k)
```

```
plot(t, s2(k,:))
```

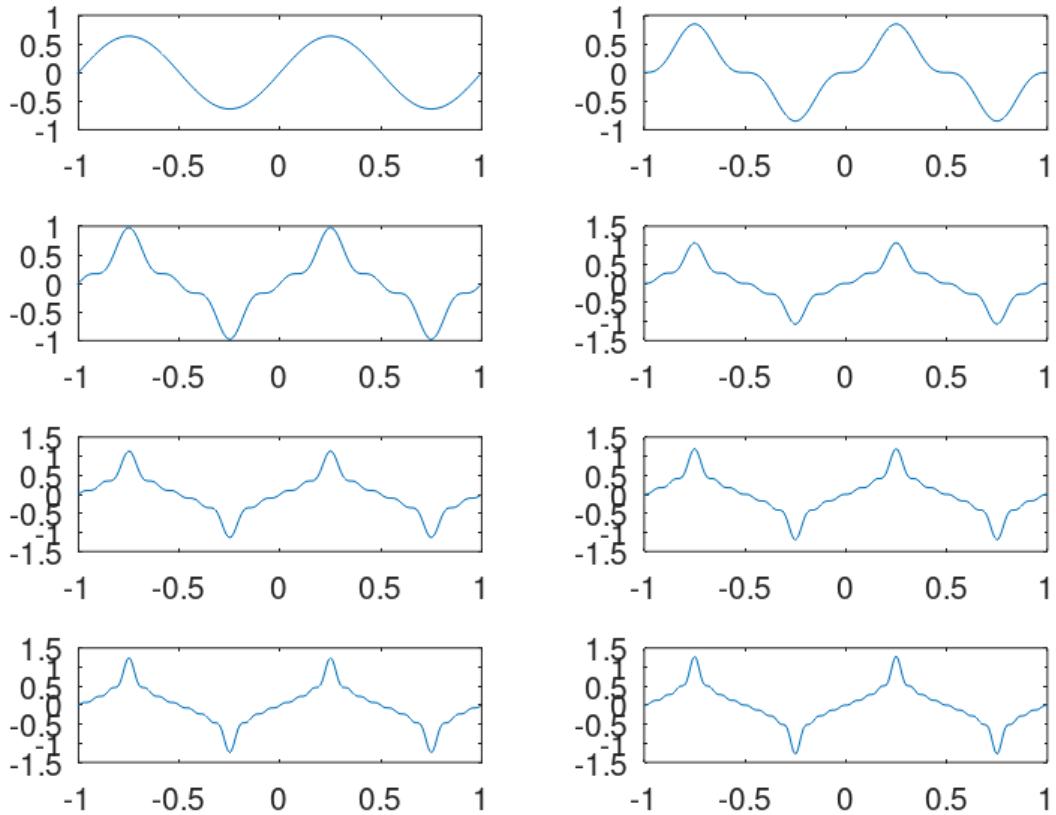
```
end
```

5. Экспортируйте полученный график в файл в формате .png.

```
#код
N=8;
t=-1:0.01:1;
A=1;
T=1;
nh=(1:N)*2-1;
Am=2/pi ./ nh;
Am(2:2:end) = -Am(2:2:end);
harmonics=cos(2 * pi * nh' * t/T);
s1=harmonics.*repmat(Am',1,Length(t));
s2=cumsum(s1);
for k=1:N
subplot(4,2,k)
plot(t, s2(k,:))
end
print("plot5.png");
```

6. Скорректируйте код для реализации меандра через синусы. Получите соответствующие графики.

```
#код
N=8;
t=-1:0.01:1;
A=1;
T=1;
nh=(1:N)*2-1;
Am=2/pi ./ nh;
Am(2:2:end) = -Am(2:2:end);
harmonics=sin(2 * pi * nh' * t/T);
s1=harmonics.*repmat(Am',1,Length(t));
s2=cumsum(s1);
for k=1:N
subplot(4,2,k)
plot(t, s2(k,:))
end
print("plot5.1.png");%чтобы сохранить графическое изображение в формате png
```



1.3.2.1.2. Графики меандра, содержащего различное число гармоник графики \sin

1.3.3. Определение спектра и параметров сигнала

1.3.3.1. Постановка задачи

1. Определить спектр двух отдельных сигналов и их суммы.

Частота дискретизации (количество отсчетов) выбирается на основе теоремы Котельникова как удвоенная ширина спектра исходного сигнала (таким образом, в следующем примере достаточно было взять частоту дискретизации 80 Гц).

2. Выполнить задание с другой частотой дискретизации. Пояснить, что будет, если взять частоту дискретизации меньше 80 Гц?

Для двух синусоидальных сигналов требуется определить их спектр.

1.3.3.2. Выполнение работы

1. В вашем рабочем каталоге создайте каталог `spectre1` и в нём новый сценарий с именем, `spectre.m`.
2. В коде созданного сценария задайте начальные значения:
3. Далее в коде задайте два синусоидальных сигнала разной частоты:
4. Постройте графики сигналов

В файле `spectre.m` задаем параметры сигналов:

```
mkdir 'signal';
mkdir 'spectre';
```

```

tmax = 0.5;% Длина сигнала (с)
fd = 512; Частота дискретизации (Гц) (количество
f1 = 10;% Частота первого сигнала (Гц)
f2 = 40;% Частота второго сигнала (Гц)
a1 = 1;% Амплитуда первого сигнала
a2 = 0.7;% Амплитуда второго сигнала
fd2 = fd/2; % Спектр сигнала
Рассмотрим два сигнала (синусоиды) разной частоты
t = 0:1./fd:tmax; Массив отсчетов времени
signal1 = a1*sin(2*pi*t*f1);
signal2 = a2*sin(2*pi*t*f2);
plot(signal1, 'b'); голубая
hold on
plot(signal2, 'r'); красная
hold off
title('Signal');
print 'signal/spectre.png';

```

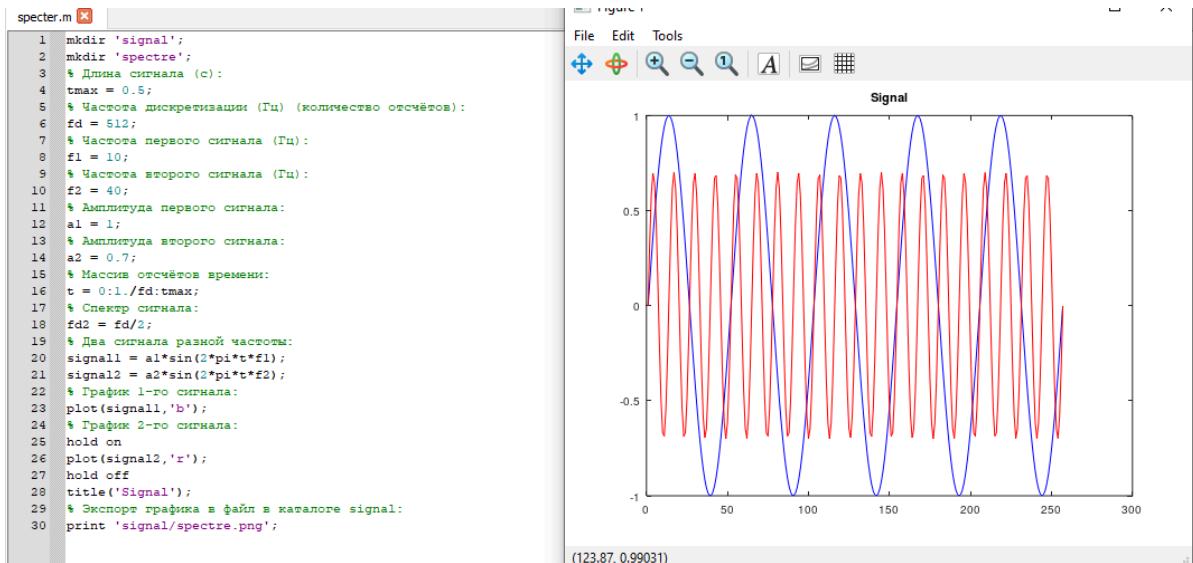


рис.1.3.3.2.1.2.3.4. Два синусоидальных сигнала разной частоты

Принимая во внимание реализацию преобразования Фурье, мы скорректируем график спектра (рис.1.3.3.2.1.2.3.4.): дублирующиеся отрицательные частоты отбрасываются, а также учитывается, что на каждом шаге вычисления быстрого преобразования Фурье амплитуды сигналов суммируются. Мы добавляем следующий код в файл spectre.m.

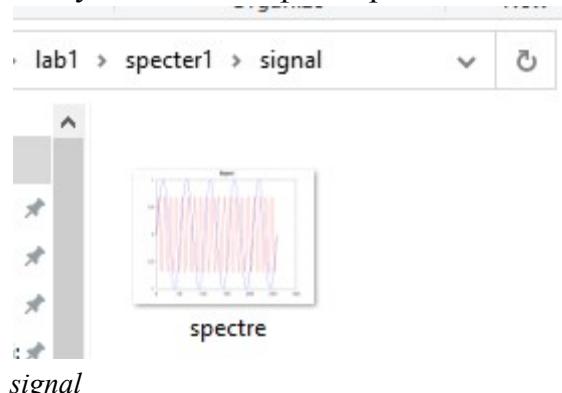


рис.1.3.3.2.4. Экспорт графика в файл в каталоге

5. С помощью быстрого преобразования Фурье найдите спектры сигналов, добавив в файл spectre.m

6. Учитывая реализацию преобразования Фурье, скорректируйте график спектра: отбросьте дублирующие отрицательные частоты, а также примите в расчёт то, что на каждом шаге вычисления быстрого преобразования Фурье происходит суммирование амплитуд сигналов. Для этого добавьте в файл spectre.m следующий код:

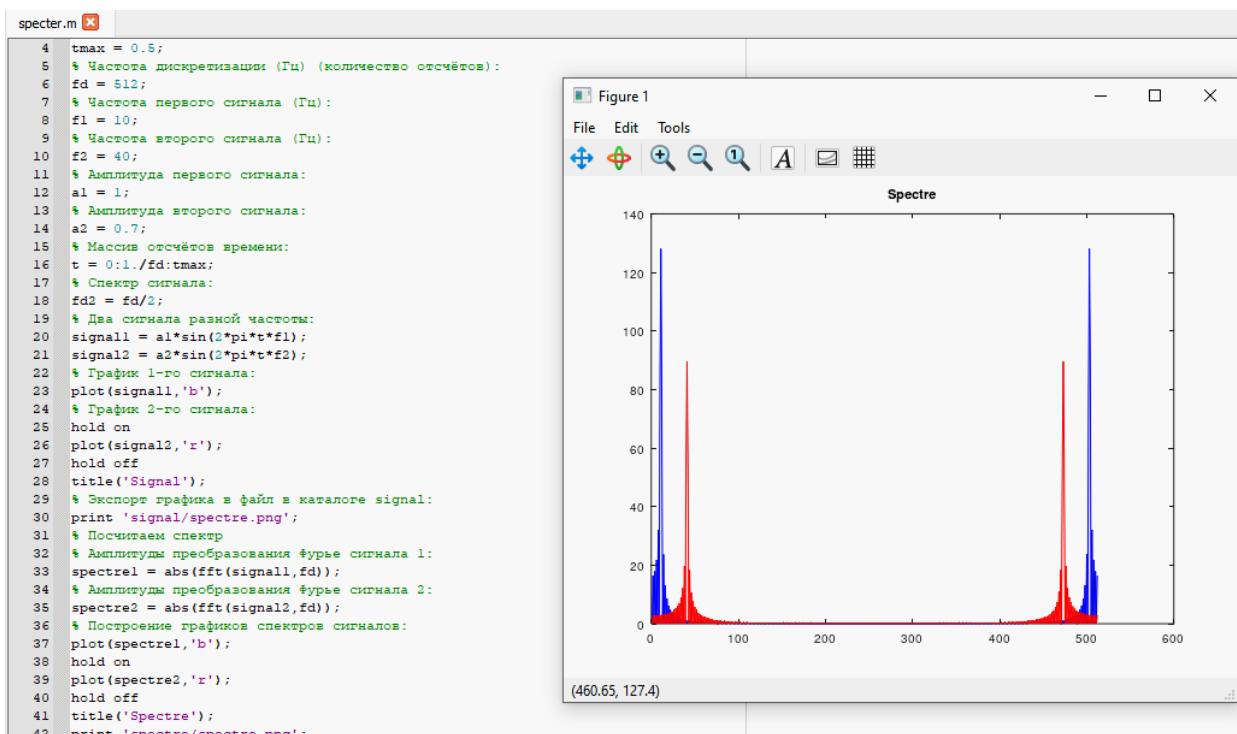


рис.1.3.3.2.5.6. График спектров синусоидальных сигналов и код

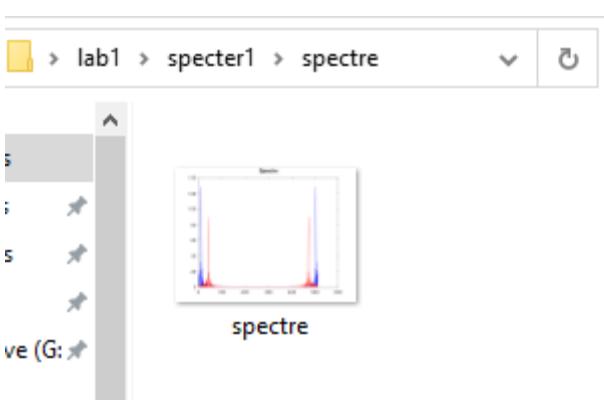


рис.1.3.3.2.5.6. графики png в spectre

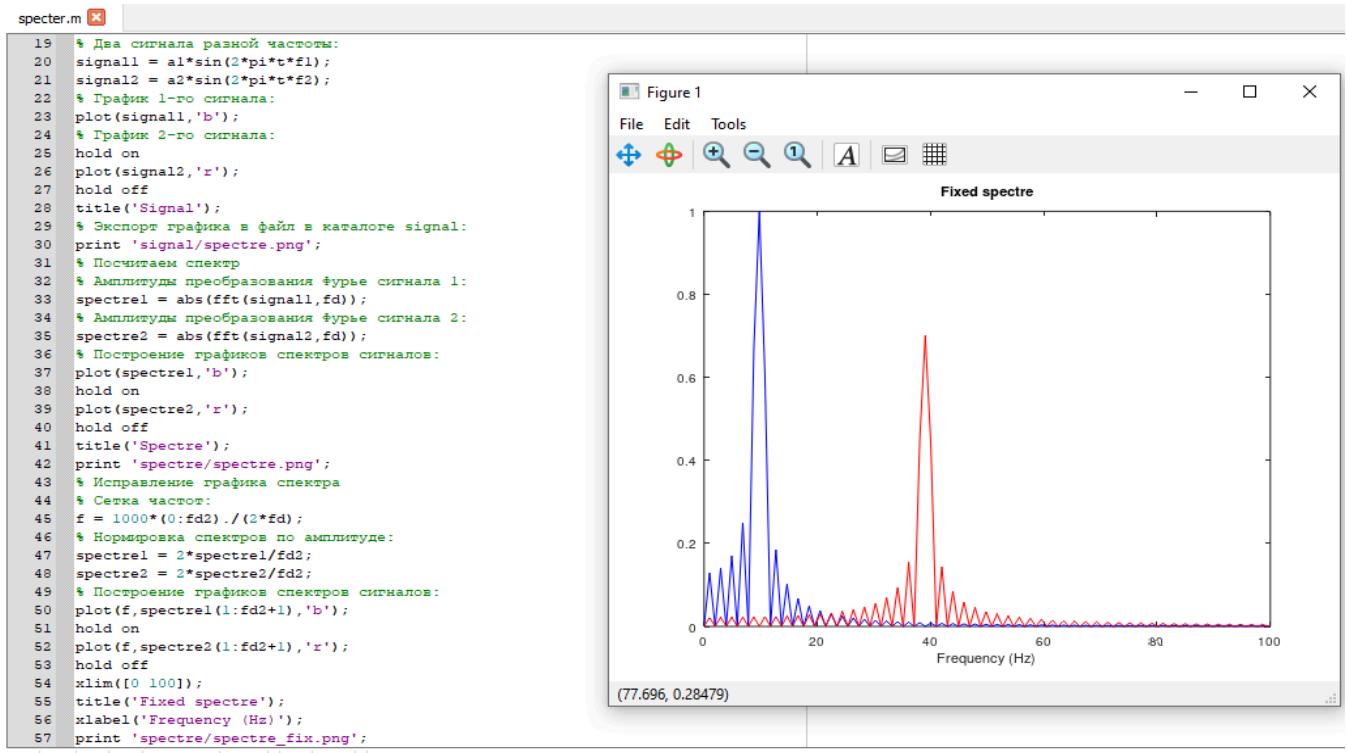


рис.1.3.3.2.6. Исправленный график спектров синусоидальных сигналов и код

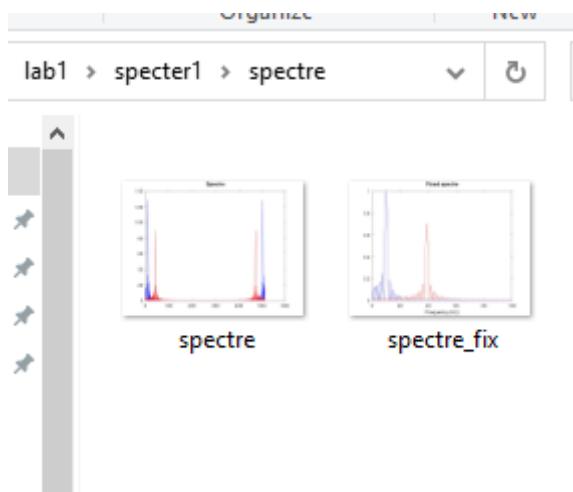


рис.1.3.3.2.6 графики png в spectre

7. Найдите спектр суммы рассмотренных сигналов, создав каталог spectr_sum и файл в нём spectre_sum.m со следующим кодом:

```

# код
mkdir 'signal';
mkdir 'spectre';
% Длина сигнала (с):
tmax = 0.5;
% Частота дискретизации (Гц) (количество отсчётов):
fd = 512;

```

```

% Частота первого сигнала (Гц):
f1 = 10;
% Частота второго сигнала (Гц):
f2 = 40;
% Амплитуда первого сигнала:
a1 = 1;
% Амплитуда второго сигнала:

a2 = 0.7;
% Спектр сигнала
fd2 = fd/2;
% Сумма двух сигналов (синусоиды) разной частоты:
% Массив отсчётов времени:
t = 0:1./fd:tmax;
signal1 = a1*sin(2*pi*t*f1);
signal2 = a2*sin(2*pi*t*f2);
signal = signal1 + signal2;
plot(signal);
title('Signal');
print 'signal/spectre_sum.png';
% Подсчет спектра:
% Амплитуды преобразования Фурье сигнала:
spectre = fft(signal,fd);
% Сетка частот
f = 1000*(0:fd2)./(2*fd);
% Нормировка спектра по амплитуде:
spectre = 2*sqrt(spectre.*conj(spectre))./fd2;
% Построение графика спектра сигнала:
plot(f,spectre(1:fd2+1))
xlim([0 100]);
title('Spectre');
xlabel('Frequency (Hz)');
print 'spectre/spectre_sum.png';

```

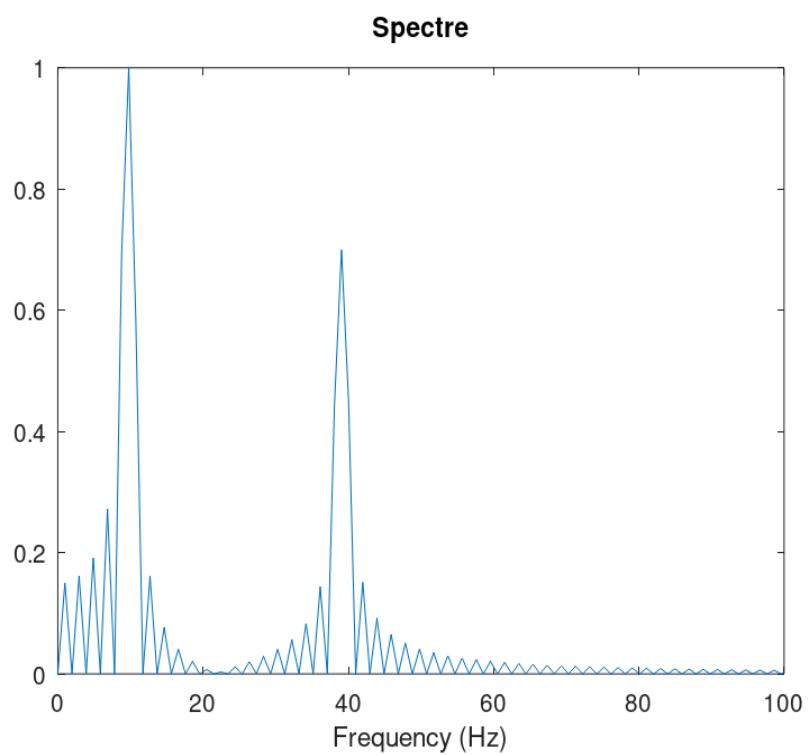


рис.1.3.3.2.7. Спектр суммарного сигнала

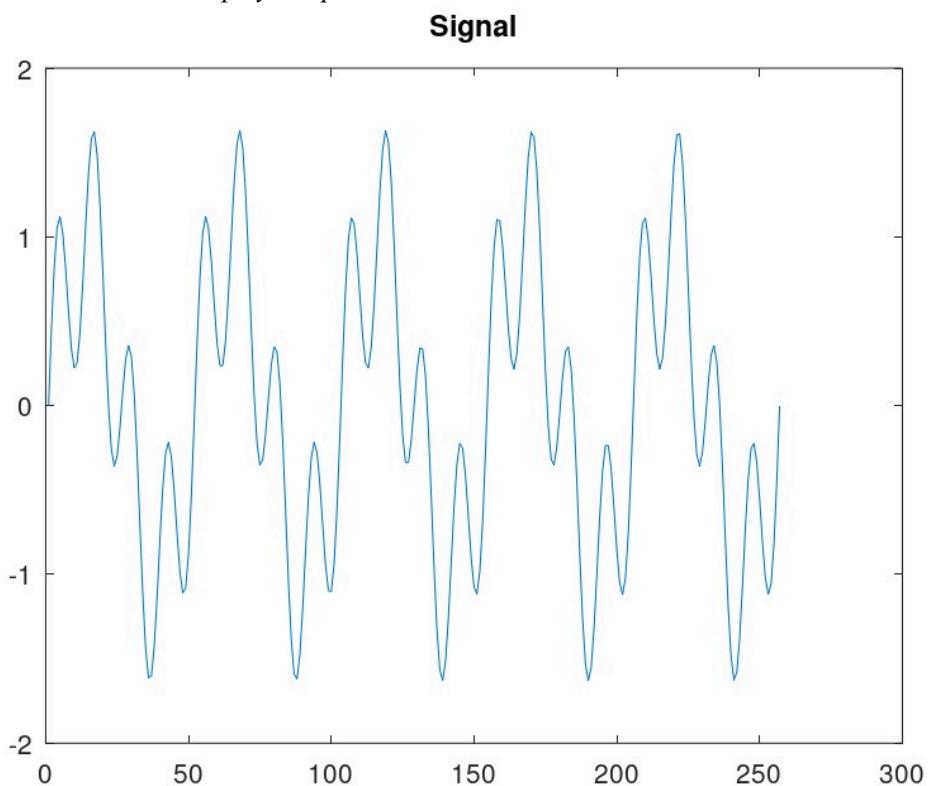


рис.1.3.3.2.7. Суммарный сигнал

1.3.4. Амплитудная модуляция

1.3.4.1. Постановка задачи

Продемонстрировать принципы модуляции сигнала на примере аналоговой амплитудной модуляции.

```
mkdir 'signal';
mkdir 'spectre';
% Модуляция синусоид с частотами 50 и 5
% Длина сигнала (с)
tmax = 0.5;
% Частота дискретизации (Гц) (количество отсчётов)
fd = 512;
% Частота сигнала (Гц)
f1 = 5;
% Частота несущей (Гц)
f2 = 50;
% Спектр сигнала
fd2 = fd/2;
% Построение графиков двух сигналов (синусоиды)
% разной частоты
% Массив отсчётов времени:
t = 0:1./fd:tmax;
signal1 = sin(2*pi*t*f1);
signal2 = sin(2*pi*t*f2);
signal = signal1 .* signal2;
plot(signal, 'b');
hold on
% Построение огибающей:
plot(signal1, 'r');
plot(-signal1, 'r');
hold off
title('Signal');
print 'signal/am.png';
% Расчет спектра:
% Амплитуды преобразования Фурье-сигнала:
spectre = fft(signal,fd);
% Сетка частот:
f = 1000*(0:fd2)./(2*fd);
% Нормировка спектра по амплитуде:
spectre = 2*sqrt(spectre.*conj(spectre))./fd2;
% Построение спектра:
plot(f,spectre(1:fd2+1), 'b')
xlim([0 100]);
title('Spectre');
xlabel('Frequency (Hz)');
print 'spectre/am.png';
```

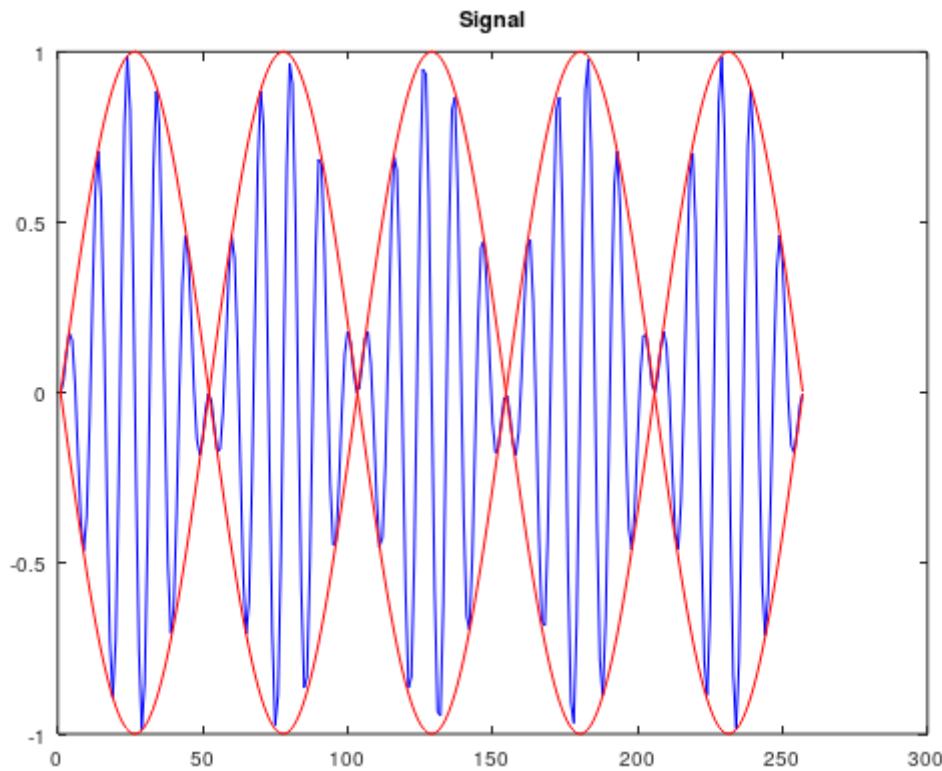


рис.1.3.3.4.1 Сигнал и огибающая при амплитудной модуляции

1.3.4.2.Выполнение работы

1. В вашем рабочем каталоге создайте каталог modulation и в нём новый сценарий с именем am.m.

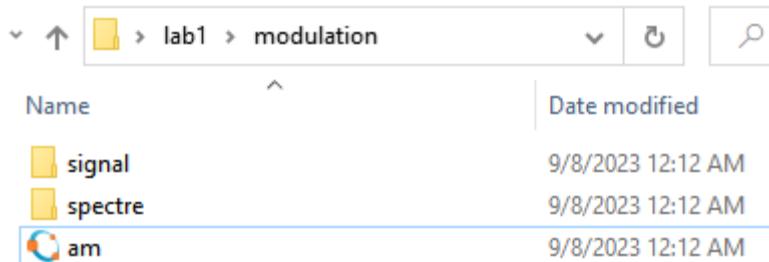


рис.1.3.4.2.1.каталог modulation и в нём новый сценарий с именем am.m.

2. Добавьте в файле am.m следующий код:

```

mkdir 'signal';
mkdir 'spectre';
% Модуляция синусоид с частотами 50 и 5
% Длина сигнала (с)
tmax = 0.5;
% Частота дискретизации (Гц) (количество отсчётов)
fd = 512;
% Частота сигнала (Гц)
f1 = 5;
% Частота несущей (Гц)
f2 = 50;
% Спектр сигнала
fd2 = fd/2;
% Построение графиков двух сигналов (синусоиды)

```

```

% разной частоты
% Массив отсчётов времени:
t = 0:1./fd:tmax;
signal1 = sin(2*pi*t*f1);
signal2 = sin(2*pi*t*f2);
signal = signal1 .* signal2;
plot(signal, 'b');
hold on
% Построение огибающей:
plot(signal1, 'r');
plot(-signal1, 'r');
hold off
title('Signal');
print 'signal/am.png';
% Расчет спектра:
% Амплитуды преобразования Фурье-сигнала:
spectre = fft(signal,fd);
% Сетка частот:
f = 1000*(0:fd2)./(2*fd);
% Нормировка спектра по амплитуде:
spectre = 2*sqrt(spectre.*conj(spectre))./fd2;
% Построение спектра:
plot(f,spectre(1:fd2+1), 'b')
xlim([0 100]);
title('Spectre');
xlabel('Frequency (Hz)');
print 'spectre/am.png';

```

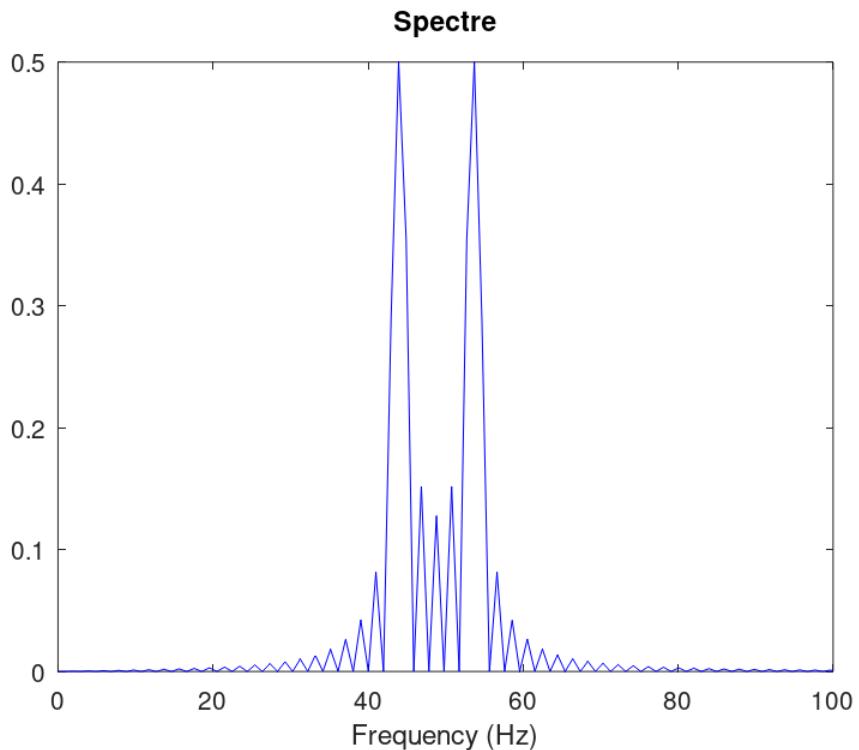


рис.1.3.4.2.1. Спектр сигнала при амплитудной модуляции

1.3.5. Кодирование сигнала. Исследование свойства самосинхронизации сигнала

1.3.5.1. Постановка задачи

По заданных битовых последовательностей требуется получить кодированные сигналы для нескольких кодов, проверить свойства самосинхрониземости кодов, получить спектры.

1.3.5.2. Выполнение работы

1. В вашем рабочем каталоге создайте каталог coding и в нём файлы main.m, maptowave.m, unipolar.m, ami.m, bipolarnrz.m, bipolarrrz.m, manchester.m, diffmanc.m, calcspctre.m.

```
>> mkdir('coding');
>> cd('coding');
>> fid = fopen('main.m', 'w');
>> fid = fopen('maptowave.m', 'w');
>> fid = fopen('unipolar.m', 'w');
>> fid = fopen('ami.m', 'w');
>> fid = fopen('bipolarnrz.m', 'w');
>> fid = fopen('bipolarrrz.m', 'w');
>> fid = fopen('manchester.m', 'w');
>> fid = fopen('diffmanc.m', 'w');
>> fid = fopen('calcspctre.m', 'w');
>> |
```

рис. 1.3.5.2.1. создание каталога и файлов

2. В окне интерпретатора команд проверьте, установлен ли у вас пакет расширений signal:

```
>> pkg list
```

Package Name	Version	Installation directory
audio	2.0.7	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
biosig	2.5.2	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
cfitsio	0.0.5	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
communications	1.2.6	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
control	3.6.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
data-smoothing	1.3.0	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
database	2.4.4	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
dataframe	1.2.0	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
dicon	0.5.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
financial	0.5.3	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
fuzzy-logic-toolkit	0.4.6	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
ga	0.10.3	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
general	2.1.2	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
generate_html	0.3.3	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
geometry	4.0.0	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
gsl	2.1.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
image	2.14.0	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
instrument-control	0.9.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
interval	3.2.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
io	2.6.4	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
linear-algebra	2.2.3	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
lssa	0.1.4	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
ltfat	2.3.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
mapping	1.4.2	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
matgeon	1.2.3	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
miscellaneous	1.3.0	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
mqtt	0.0.4	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
nan	3.7.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
netcdf	1.0.16	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
nurbs	1.4.3	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
ocs	0.1.5	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
octproj	3.0.2	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
optim	1.6.2	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
optiminterp	0.3.7	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
quaternion	2.4.0	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
queueing	1.2.7	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
signal	1.4.5	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
sockets	1.4.1	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav
sparsersb	1.0.9	C:\Users\Hp\AppData\Local\Programs\GNU Octave\Octav

рис. 1.3.5.2.2. pkg list

3. В файле main.m подключите пакет signal и задайте входные кодовые последовательности:

```

% coding/main.m
% Подключение пакета signal:
pkg load signal;
% Входная кодовая последовательность:
data=[0 1 0 0 1 1 0 0 0 1 1 0];
% Входная кодовая последовательность для проверки свойства
% самосинхронизации:
data_sync=[0 0 0 0 0 0 1 1 1 1 1 1];
% Входная кодовая последовательность для построения спектра сигнала:
data_spectre=[0 1 0 1 0 1 0 1 0 1 0 1];
% Создание каталогов signal, sync и spectre для размещения графиков:
mkdir 'signal';
mkdir 'sync';
mkdir 'spectre';
axis("auto");
% Униполярное кодирование
wave=unipolar(data);
plot(wave);
ylim([-1 6]);
title('Unipolar');
print 'signal/unipolar.png';
% Кодирование ami
wave=ami(data);
plot(wave)
title('AMI');
print 'signal/ami.png';
% Кодирование NRZ
wave=bipolarnrz(data);
plot(wave);
title('Bipolar Non-Return to Zero');
print 'signal/bipolarnrz.png';
% Кодирование RZ
wave=bipolarrz(data);
plot(wave)
title('Bipolar Return to Zero');
print 'signal/bipolarrz.png';
% Манчестерское кодирование
wave=manchester(data);
plot(wave)
title('Manchester');
print 'signal/manchester.png';
% Дифференциальное манчестерское кодирование
wave=diffmanc(data);
plot(wave)
title('Differential Manchester');
print 'signal/diffmanc.png';
% Униполярное кодирование
wave=unipolar(data_sync);
plot(wave);
ylim([-1 6]);

```

```
title('Unipolar');
print 'sync/unipolar.png';
% Кодирование AMI
wave=ami(data_sync);
plot(wave)
title('AMI');
print 'sync/ami.png';
% Кодирование NRZ
wave=bipolarnrz(data_sync);
plot(wave);
title('Bipolar Non-Return to Zero');
print 'sync/bipolarnrz.png';
% Кодирование RZ
wave=bipolarrz(data_sync);
plot(wave)
title('Bipolar Return to Zero');
print 'sync/bipolarrz.png';
% Манчестерское кодирование
wave=manchester(data_sync);
plot(wave)
title('Manchester');
print 'sync/manchester.png';
% Дифференциальное манчестерское кодирование
wave=diffmanc(data_sync);
plot(wave)
title('Differential Manchester');
print 'sync/diffmanc.png';
% Униполярное кодирование:
wave=unipolar(data_spectre);
spectre=calcspectre(wave);
title('Unipolar');
print 'spectre/unipolar.png';
% Кодирование AMI:
wave=ami(data_spectre);
spectre=calcspectre(wave);
title('AMI');
print 'spectre/ami.png';
% Кодирование NRZ:
wave=bipolarnrz(data_spectre);
spectre=calcspectre(wave);
title('Bipolar Non-Return to Zero');
print 'spectre/bipolarnrz.png';
% Кодирование RZ:
wave=bipolarrz(data_spectre);
spectre=calcspectre(wave);
title('Bipolar Return to Zero');
print 'spectre/bipolarrz.png';
% Манчестерское кодирование:
wave=manchester(data_spectre);
spectre=calcspectre(wave);
title('Manchester');
```

```

print 'spectre/manchester.png';
% Дифференциальное манчестерское кодирование:
wave=diffmanc(data_spectre);
spectre=calcspectre(wave);
title('Differential Manchester');
print 'spectre/diffmanc.png';

```

4. В файле maptowave.m пропишите функцию, которая по входному битовому потоку строит график сигнала:

```

% coding/maptowave.m
function wave=maptowave(data)
data=upsample(data,100);
wave=filter(5*ones(1,100),1,data);

```

5. В файлах unipolar.m, ami.m, bipolarnrz.m, bipolarrz.m, manchester.m, diffmanc.m пропишите соответствующие функции преобразования кодовой последовательности data с вызовом функции maptowave для построения соответствующего графика.

```

%unipolar.m
% coding/unipolar.m
% Униполярное кодирование:
function wave=unipolar(data)
wave=maptowave(data);

%ami.m
% coding/ami.m
% Кодирование AMI:
function wave=ami(data)
am=mod(1:length(data(data==1)),2);
am(am==0)=-1;
data(data==1)=am;
wave=maptowave(data);

```

```

% bipolarnrz.m
% coding/bipolarnrz.m
% Кодирование NRZ:
function wave=bipolarnrz(data)
data(data==0)=-1;
wave=maptowave(data);

```

```

%bipolarrz.m
% coding/bipolarrz.m
% Кодирование RZ:
function wave=bipolarrz(data)
data(data==0)=-1;
data=upsample(data,2);

```

```

wave=maptowave(data);

% manchester.m
% coding/manchester.m
% Манчестерское кодирование:
function wave=manchester(data)
data(data==0)=-1;
data=upsample(data,2);
data=filter([-1 1],1,data);
wave=maptowave(data);

%diffmanc.m
% coding/diffmanc.m
% Дифференциальное манчестерское кодирование
function wave=diffmanc(data)
data=filter(1,[1 1],data);
data=mod(data,2);
wave=manchester(data);

```

6. В файле calcspectre.m пропишите функцию построения спектра сигнала
 $\% calcspectre.m$

$\% \text{Функция построения спектра сигнала:}$
 $\text{function spectre} = \text{calcspectre}(wave)$
 $\% \text{Частота дискретизации (Гц):}$
 $Fd = 512;$
 $Fd2 = Fd/2;$
 $Fd3 = Fd/2 + 1;$
 $X = fft(wave,Fd);$
 $spectre = X.*conj(X)/Fd;$
 $f = 1000*(0:Fd2)/Fd;$
 $plot(f,spectre(1:Fc3));$
 $xlabel('Frequency (Hz)');$

7. Запустите главный скрипт main.m. В каталоге signal должны быть получены файлы с графиками кодированного сигнала , в каталоге sync — файлы с графиками, иллюстрирующими свойства самосинхронизации , в каталоге spectre — файлы с графиками спектров сигналов.

в каталоге sync

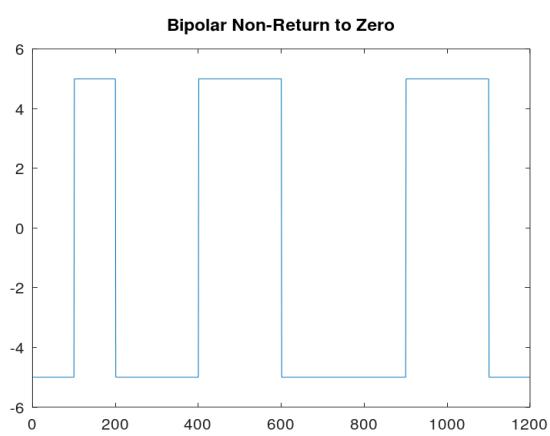


рис.1. Униполярное кодирование

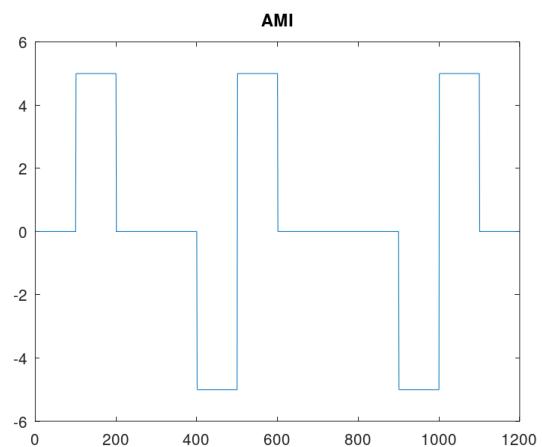


рис.2. ami

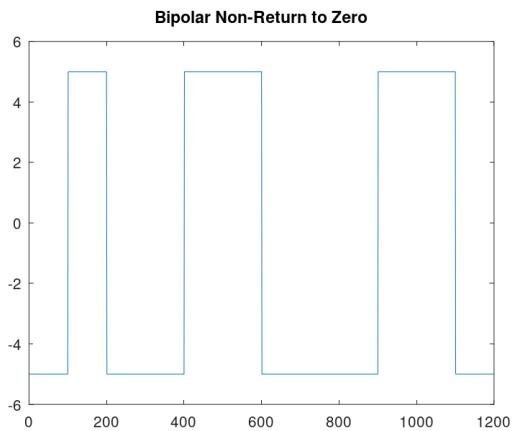


рис.3.NRZ

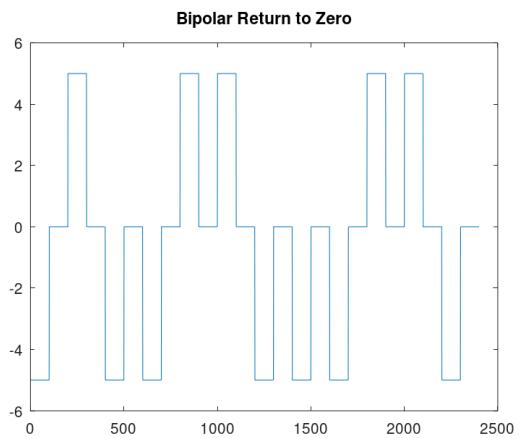


рис.4. RZ

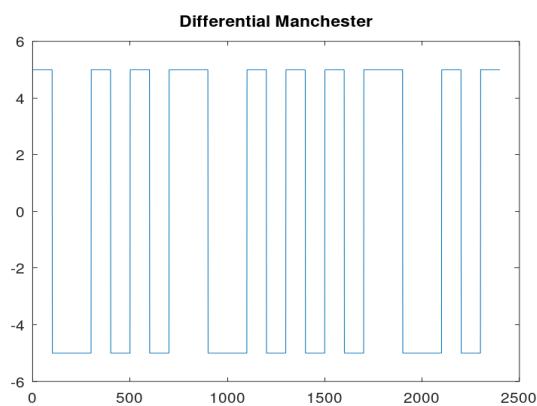
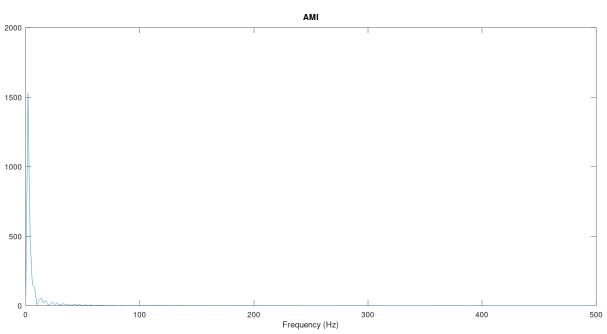


рис.5. Диффер. манчестерское

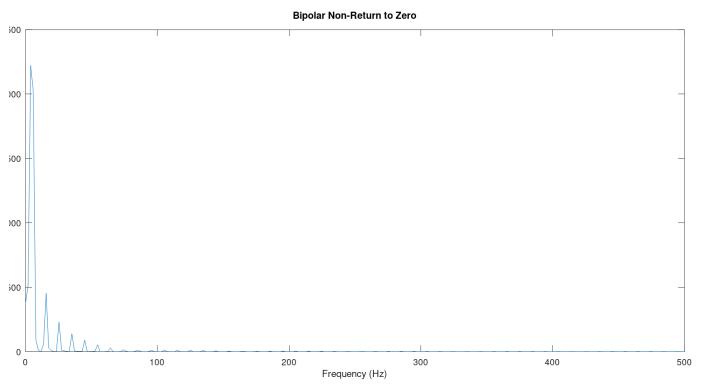


рис.6 Манчестерское

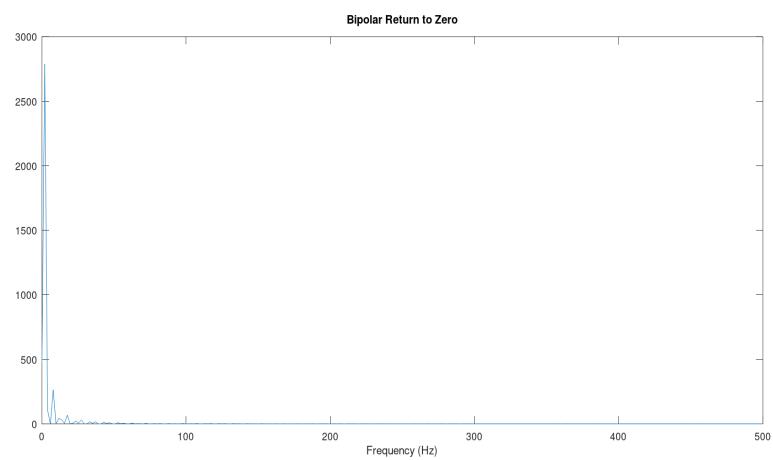
в каталоге spectre



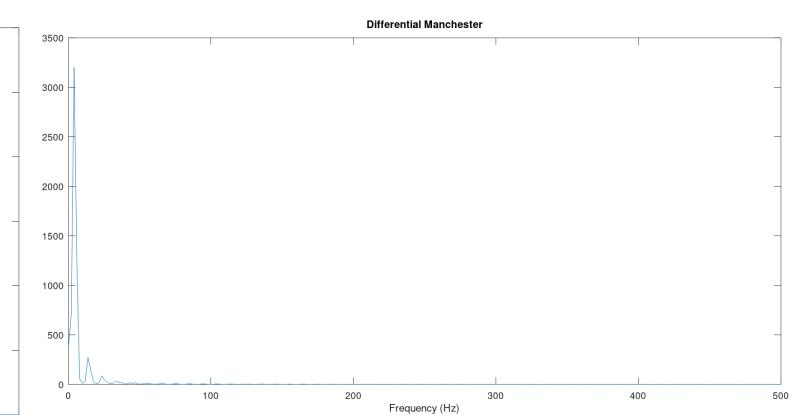
puc.1 ami



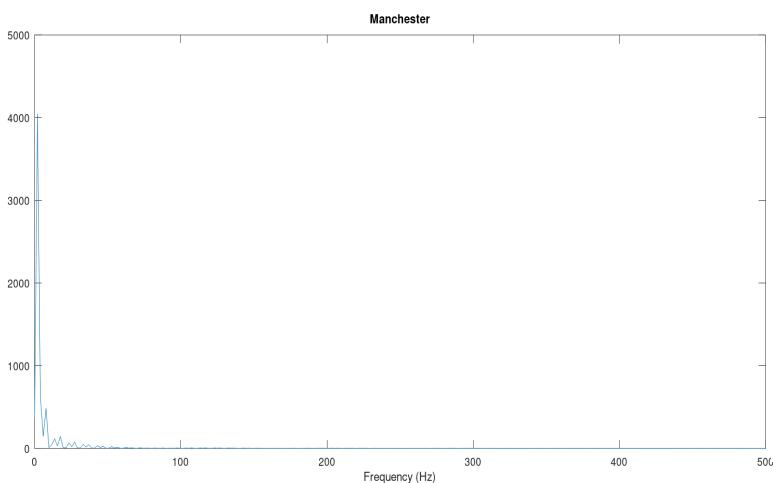
puc.2 bipolarnz



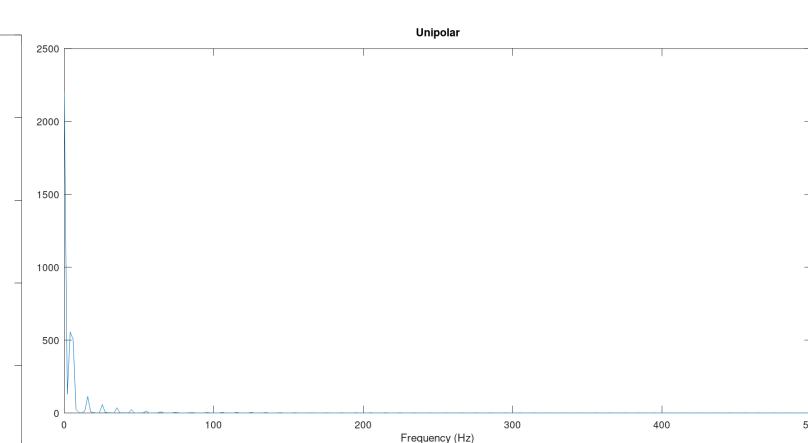
puc.3 bipolarrz



puc.4 diffmanc

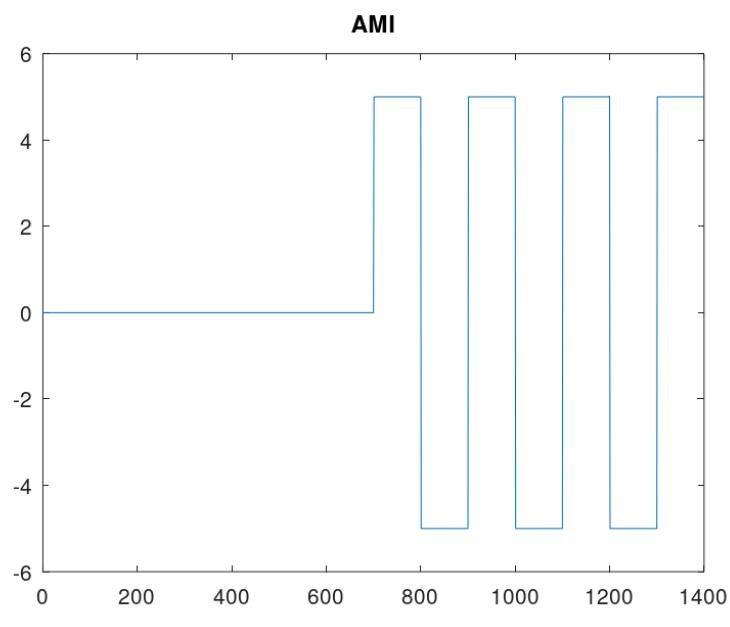


puc.5 machester

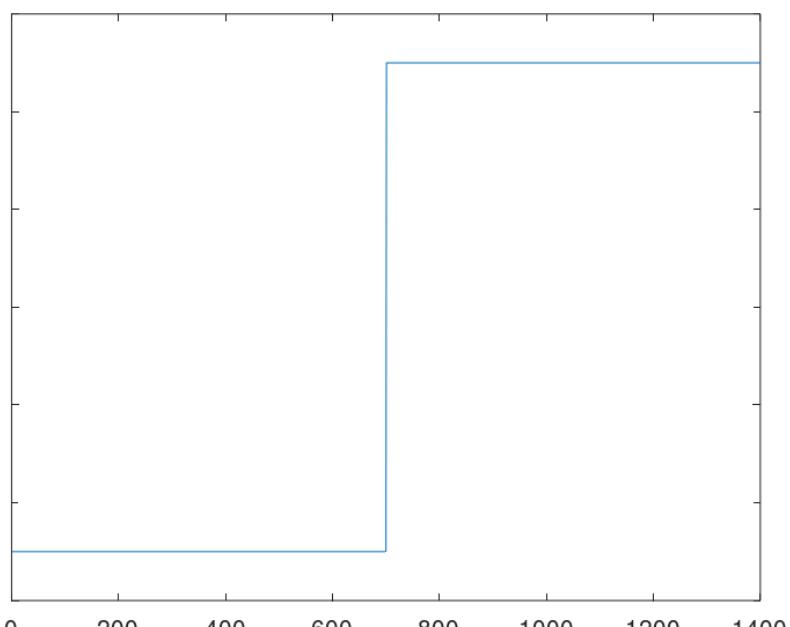


puc.6 unipolar

в каталоге sync



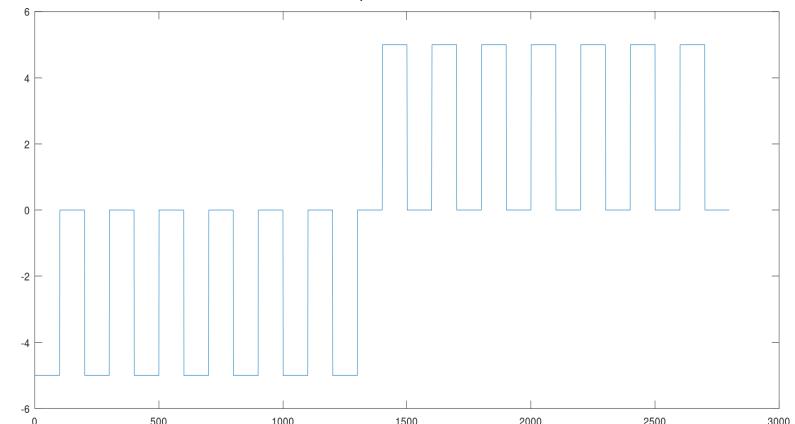
Bipolar Non-Return to Zero



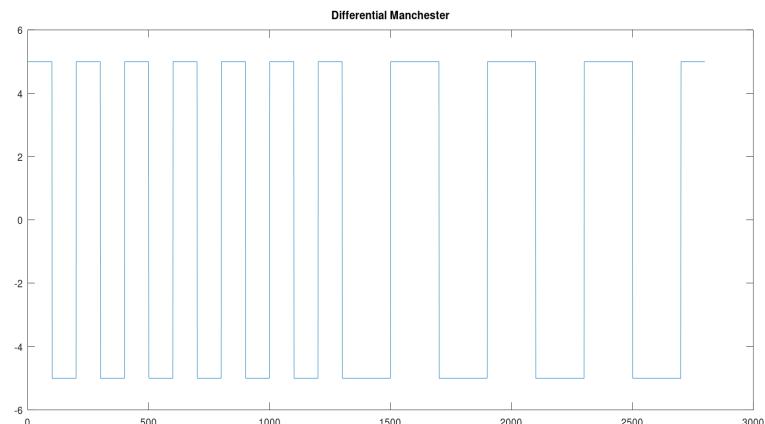
puc.1 ami

puc.2 bipolarnz

Bipolar Return to Zero



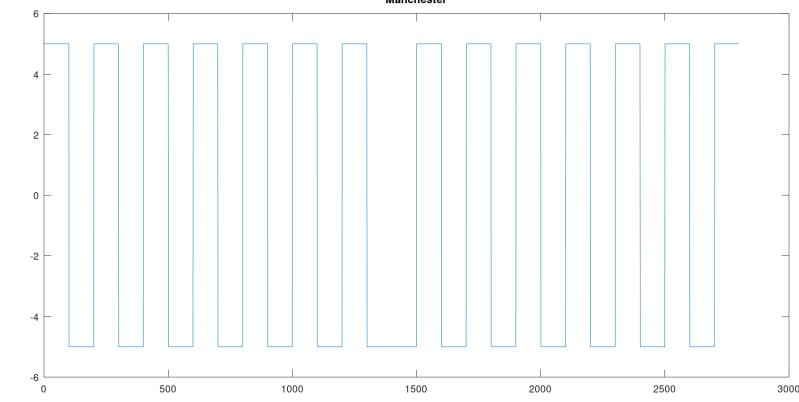
Differential Manchester



puc.3 bipolarrz

puc.4 diffmanc

Manchester



Unipolar

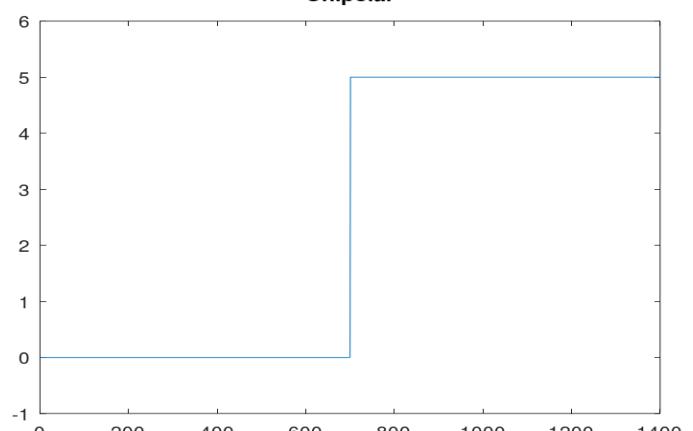


рис.5 manchester

рис.6 unipolar

Выводы

В этой лабораторной работе мы изучали методы кодирования и модуляции сигналов с использованием высокоуровневого языка программирования Octave. Мы определили спектр и параметры сигнала и показали принципы модуляции сигнала на примере аналоговой амплитудной модуляции.