

Лабораторная работа №1

Дисциплина: Сетевые технологии

Жибицкая Евгения Дмитриевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выводы	22
	Список литературы	23

Список иллюстраций

2.1	Установка Octave	6
2.2	Создание сценария	7
2.3	Код для sin_plot	7
2.4	График sin_plot	8
2.5	Код для sin_cos_plot	9
2.6	График sin_cos_plot	10
2.7	Код для meandr.m	11
2.8	Код для meandr.m через синус	12
2.9	График meandr.m	13
2.10	Создание каталога и файла	14
2.11	График spectre.m	14
2.12	Доработка кода	15
2.13	Исправленный график	16
2.14	Суммарный сигнал	16
2.15	Спектр суммарного сигнала	17
2.16	Спектр сигнала при амплитудной модуляции	18
2.17	Подготовка рабочего пространства	18
2.18	Функции	20
2.19	Функции	21
2.20	Графики	21
2.21	Графики	21
2.22	Графики	21

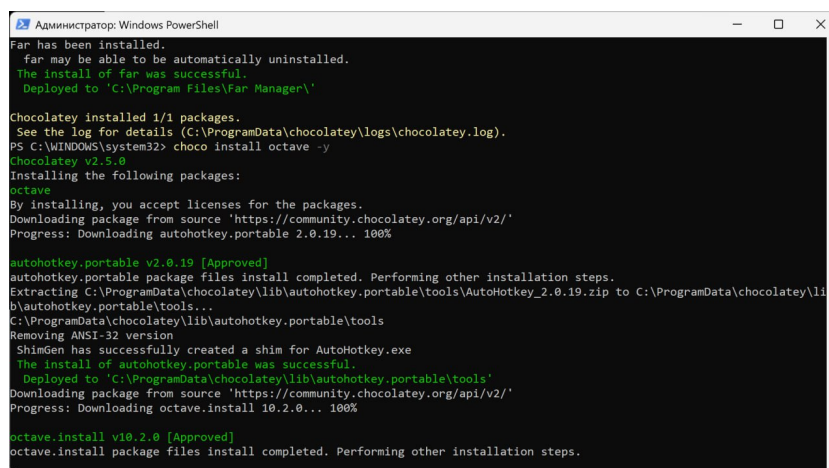
Список таблиц

1 Цель работы

Знакомство с Octave, получение навыков по работе с ним. Изучение методов кодирования и модуляции сигналов с помощью высокоуровневого языка программирования Octave. Определение спектра и параметров сигнала. Демонстрация принципов модуляции сигнала на примере аналоговой амплитудной модуляции. Исследование свойства самосинхронизации сигнала

2 Выполнение лабораторной работы

Для выполнения работы устанавливаем Octave с помощью Chocolatey(рис. 2.1).



```
Администратор: Windows PowerShell
Far has been installed.
  far may be able to be automatically uninstalled.
  The install of far was successful.
  Deployed to 'C:\Program Files\Far Manager\'

Chocolatey installed 1/1 packages.
  See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\WINDOWS\system32> choco install octave -y
Chocolatey v2.5.0
Installing the following packages:
octave
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading autohotkey.portable 2.0.19... 100%

autohotkey.portable v2.0.19 [Approved]
autohotkey.portable package files install completed. Performing other installation steps.
Extracting C:\ProgramData\chocolatey\lib\autohotkey.portable\tools\AutoHotkey_2.0.19.zip to C:\ProgramData\chocolatey\lib\autohotkey.portable\tools...
C:\ProgramData\chocolatey\lib\autohotkey.portable\tools
Removing ANSI-32 version
ShimGen has successfully created a shim for AutoHotkey.exe
The install of autohotkey.portable was successful.
  Deployed to 'C:\ProgramData\chocolatey\lib\autohotkey.portable\tools'
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading octave.install 10.2.0... 100%

octave.install v10.2.0 [Approved]
octave.install package files install completed. Performing other installation steps.
```

Рис. 2.1: Установка Octave

Запускаем Octave GUI, переходим в редактор и создаем новый сценарий plot_sin.m(рис. 2.2).

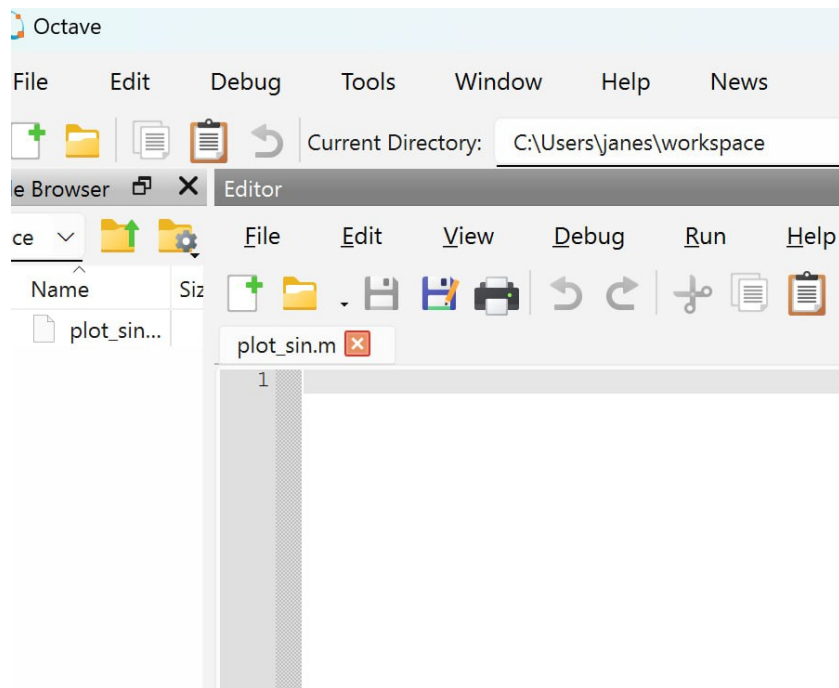


Рис. 2.2: Создание сценария

Далее добавляем туда листинг(рис. 2.3) и запускаем сценарий, получаем график(рис. 2.4).

```

plot_sin.m
1 % Формирование массива x:
2 x=-10:0.1:10;
3 % Формирование массива y.
4 y1=sin(x)+1/3*sin(3*x)+1/5*sin(5*x);
5 % Построение графика функции:
6 plot(x,y1, "-ok; y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);", "markersi:
7 % Отображение сетки на графике
8 grid on;
9 % Подпись оси X:
10 xlabel('x');
11 % Подпись оси Y:
12 ylabel('y');
13 % Название графика:
14 title('y1=sin x+ (1/3)sin(3x)+(1/5)sin(5x)');
15 % Экспорт рисунка в файл .eps:
16 print ("plot-sin.eps", "-mono", "-FArial:16", "-deps")
17 % Экспорт рисунка в файл .png:
18 print ("plot-sin.png");
19

```

Рис. 2.3: Код для sin_plot

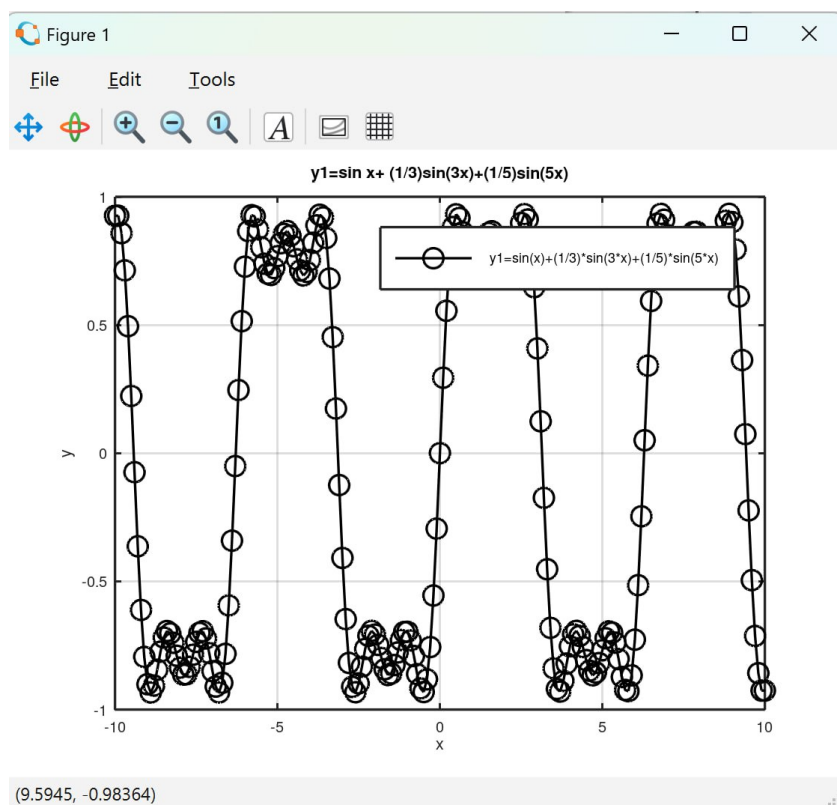


Рис. 2.4: График sin_plot

Сохраняем сценарий, поменяв ему имя, и строим теперь 2 графика - для синуса и косинуса (рис. 2.5) и (рис. 2.6).


```

plot_sin_cos.m
1 % Формирование массива x:
2 x=-10:0.1:10;
3 % Формирование массива y.
4 y1=sin(x)+1/3*sin(3*x)+1/5*sin(5*x);
5 y2=cos(x)+1/3*cos(3*x)+1/5*cos(5*x);
6 % Построение графика функции:
7 plot(x,y1, "-ok; y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);", "markersi:
8 hold on;
9 plot(x,y2, "-k; y2=cos(x)+1/3*cos(3*x)+1/5*cos(5*x);", "markersize", 4)
10 % Отображение сетки на графике
11 grid on;
12 % Подпись оси X:
13 xlabel('x');
14 % Подпись оси Y:
15 ylabel('y');
16 % Название графика:
17 title('y1=sin x+ (1/3)sin(3x)+(1/5)sin(5x), y2=cos(x)+1/3*cos(3*x)+1,
18 % Экспорт рисунка в файл .eps:
19 print("plot-sin_cos.eps", "-mono", "-FArial:16", "-deps")
20 % Экспорт рисунка в файл .png:
21 print("plot-sin_cos.png");
22

```

Рис. 2.5: Код для sin_cos_plot

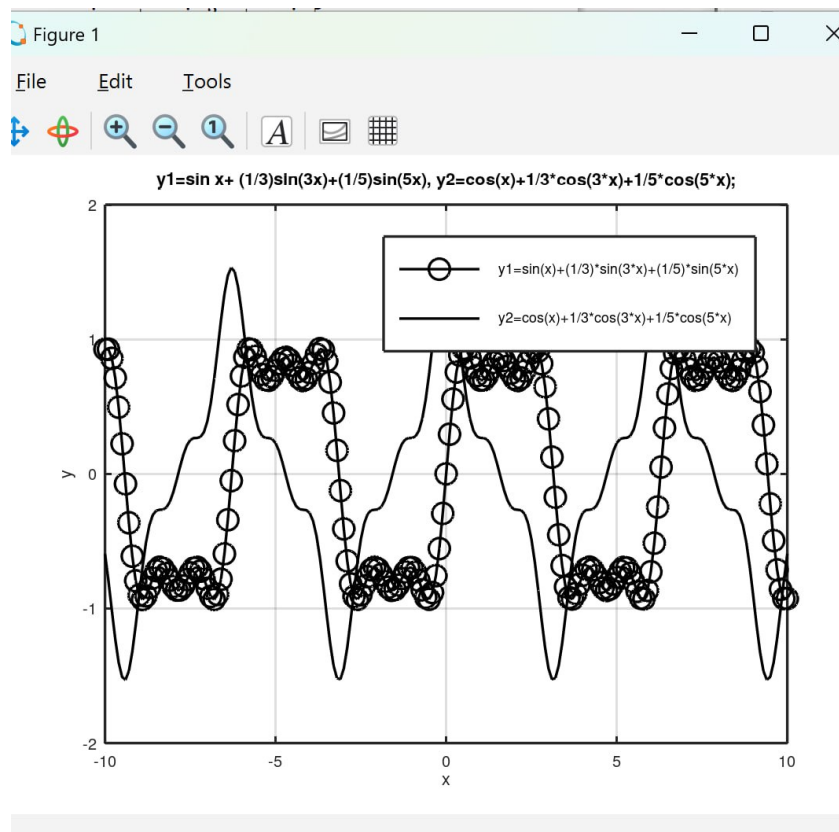


Рис. 2.6: График sin_cos_plot

Затем перейдем к разложению импульсного сигнала в частичный ряд Фурье. Создаем новый сценарий meandr.m, вставляем код(рис. 2.7). Аналогично пишем через синус, результат получаем тот же (рис. 2.8).

```

meandr.m
1 % meandr.m
2 % количество отсчетов (гармоник):
3 N=8;
4 % частота дискретизации:
5 t=-1:0.01:1;
6 % значение амплитуды:
7 A=1;
8 % период:
9 T=1;
10 % амплитуда гармоник
11 nh=(1:N)*2-1;
12 % массив коэффициентов для ряда, заданного через cos:
13 Am=2/pi ./ nh;
14 Am(2:2:end) = -Am(2:2:end);
15 % массив гармоник:
16 harmonics=cos(2 * pi * nh' * t/T);
17 % массив элементов ряда:
18 s1=harmonics.*repmat(Am',1,length(t));
19 % Суммирование ряда:
20 s2=cumsum(s1);
21 % Построение графиков:
22 for k=1:N
23     subplot(4,2,k)
24     plot(t, s2(k,:))
25 end
26 % Экспорт рисунка в файл .png:
27 print("plot-meandr.png");
28

```

Рис. 2.7: Код для meandr.m

```

1 % meandr.m
2 % количество отсчетов (гармоник):
3 N=8;
4 % частота дискретизации:
5 t=-1:0.01:1;
6 % значение амплитуды:
7 A=1;
8 % период:
9 T=1;
10 % амплитуда гармоник
11 nh=(1:N)*2-1;
12 % массив коэффициентов для ряда, заданного через sin:
13 Am=2/pi ./ nh;
14 Am(2:2:end) = Am(2:2:end);
15 % массив гармоник:
16 harmonics=sin(2 * pi * nh' * t/T);
17 % массив элементов ряда:
18 s1=harmonics.*repmat(Am',1,length(t));
19 % Суммирование ряда:
20 s2=cumsum(s1);
21 % Построение графиков:
22 for k=1:N
23     subplot(4,2,k)
24     plot(t, s2(k,:))
25 end
26 % Экспорт рисунка в файл .png:
27 print("plot-meandr_sin.png");
28

```

Рис. 2.8: Код для meandr.m через синус

Смотрим на получившийся результат(рис. 2.9).

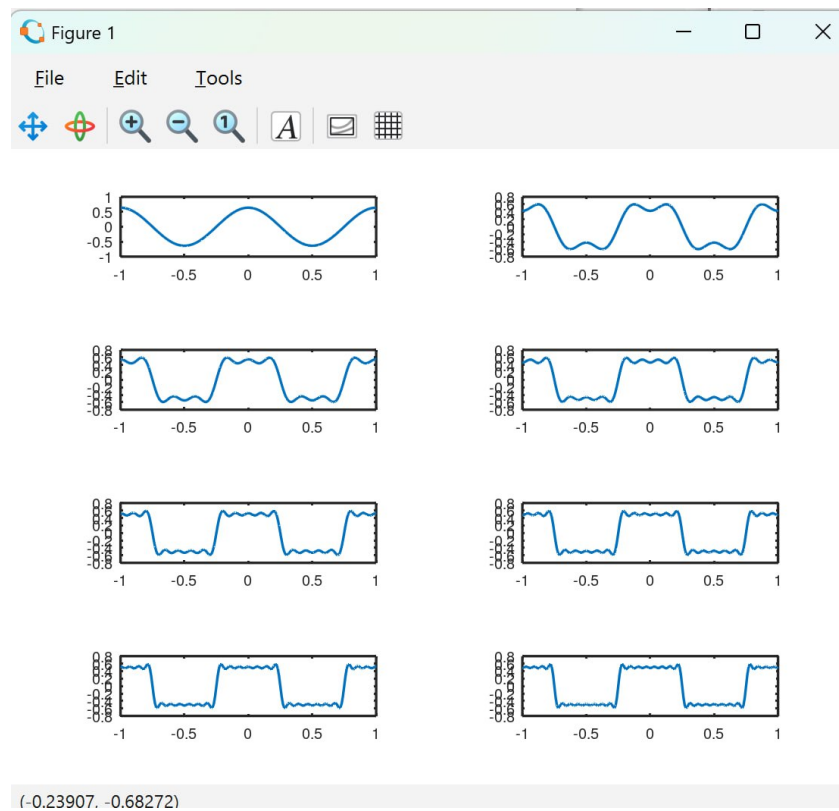


Рис. 2.9: График meandr.m

Далее определим пектр и параметры сигнала.

Создаем каталог spectre1 и в нем файл spectre.m(рис. 2.10).

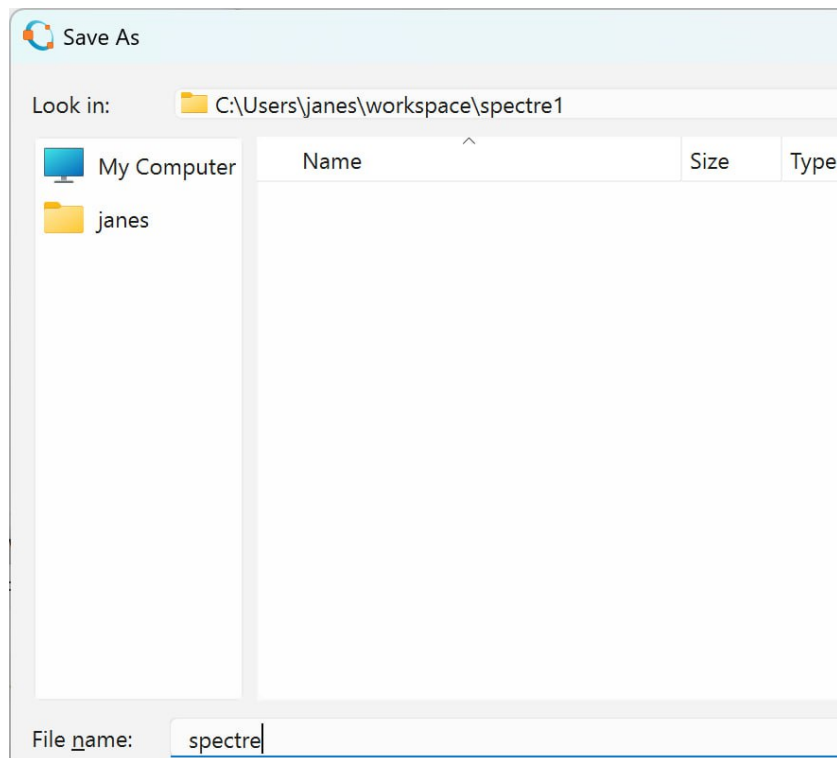


Рис. 2.10: Создание каталога и файла

Прописываем код и смотрим на результат(рис. 2.11).

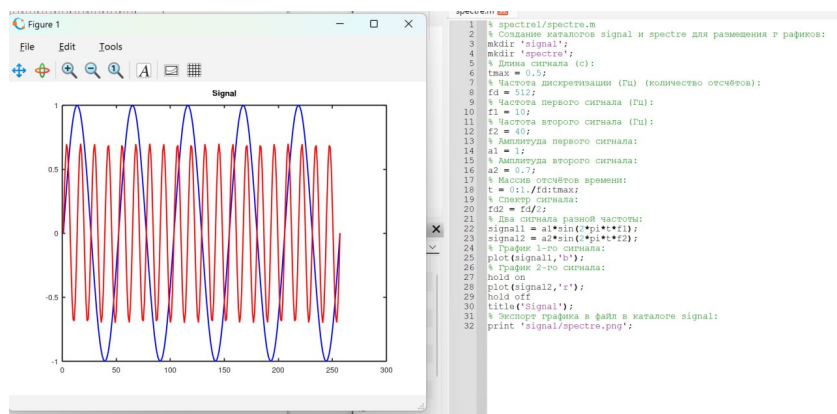


Рис. 2.11: График spectre.m

Дорабатываем код(рис. 2.12), корректируем график, отбрасывая дублирующие отрицательные частоты и принимая в расчёт то, что на каждом шаге вычисления быстрого преобразования Фурье происходит суммирование амплитуд сигналов и получаем следующий его вид(рис. 2.13).

```
спэктре.m
18 t = 0:1./fd:tmax;
19 % Спектр сигнала:
20 fd2 = fd/2;
21 % Два сигнала разной частоты:
22 signal1 = a1*sin(2*pi*t*f1);
23 signal2 = a2*sin(2*pi*t*f2);
24 % График 1-го сигнала:
25 plot(signal1,'b');
26 % График 2-го сигнала:
27 hold on
28 plot(signal2,'r');
29 hold off
30 title('Signal');
31 % Экспорт графика в файл в каталоге signal:
32 print 'signal/spectre.png';
33 % Посчитаем спектр
34 % Амплитуды преобразования Фурье сигнала 1:
35 spectre1 = abs(fft(signal1,fd));
36 % Амплитуды преобразования Фурье сигнала 2:
37 spectre2 = abs(fft(signal2,fd));
38 % Построение графиков спектров сигналов:
39 plot(spectre1,'b');
40 hold on
41 plot(spectre2,'r');
42 hold off
43 title('Spectre');
44 print 'spectre/spectre.png';
45 % Исправление графика спектра
46 % Сетка частот:
47 f = 1000*(0:fd2)./(2*fd);
48 % Нормировка спектров по амплитуде:
49 spectre1 = 2*spectre1/fd2;
50 spectre2 = 2*spectre2/fd2;
51 % Построение графиков спектров сигналов:
52 plot(f,spectre1(1:fd2+1),'b');
53 hold on
54 plot(f,spectre2(1:fd2+1),'r');
55 hold off
56 xlim([0 100]);
57 title('Fixed spectre');
58 xlabel('Frequency (Hz)');
59 print 'spectre/spectre_fix.png';
60
```

Рис. 2.12: Доработка кода

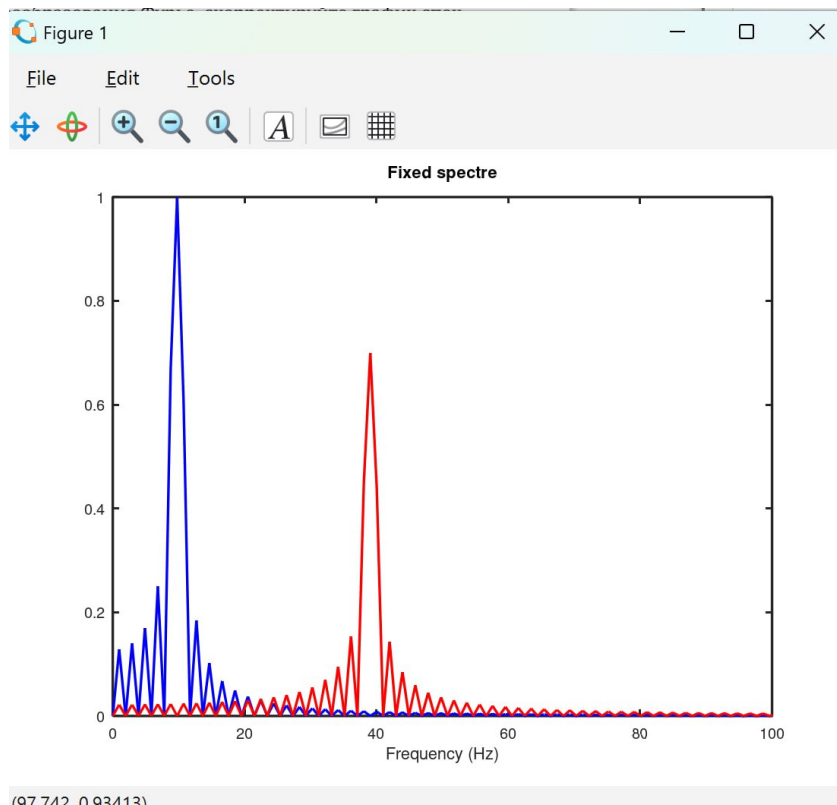


Рис. 2.13: Исправленный график

Найдем спектр суммы рассмотренных сигналов. Создаем каталог spectr_sum, файл spectre_sum с данным кодом, также полчаем спектр суммарного сигнала (рис. 2.14) и (рис. 2.15).

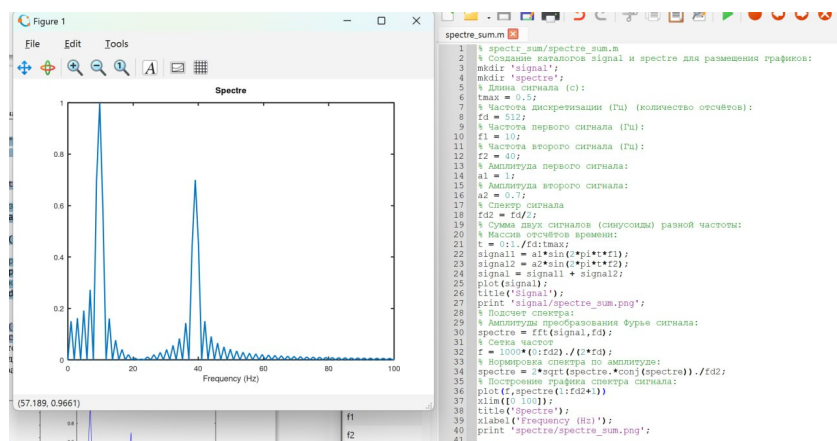


Рис. 2.14: Суммарный сигнал

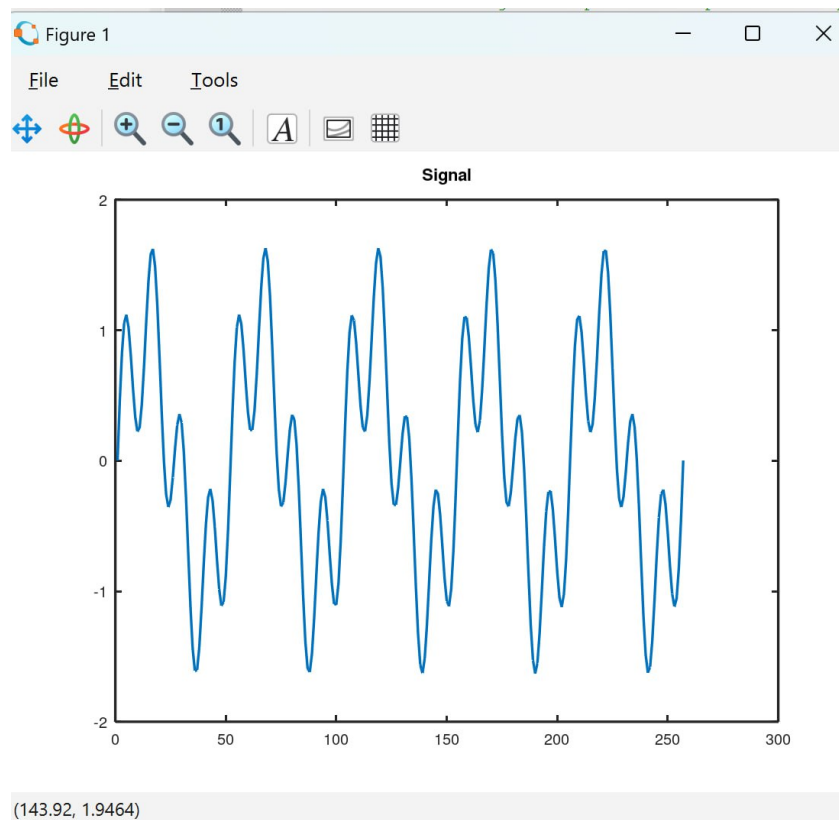


Рис. 2.15: Спектр суммарного сигнала

Ознакомимся с амплитудной модуляцией, создадим каталог `modulation` со сценарием `am.m`, увидим, что спектр произведения представляет собой свертку спектров(рис. 2.16)

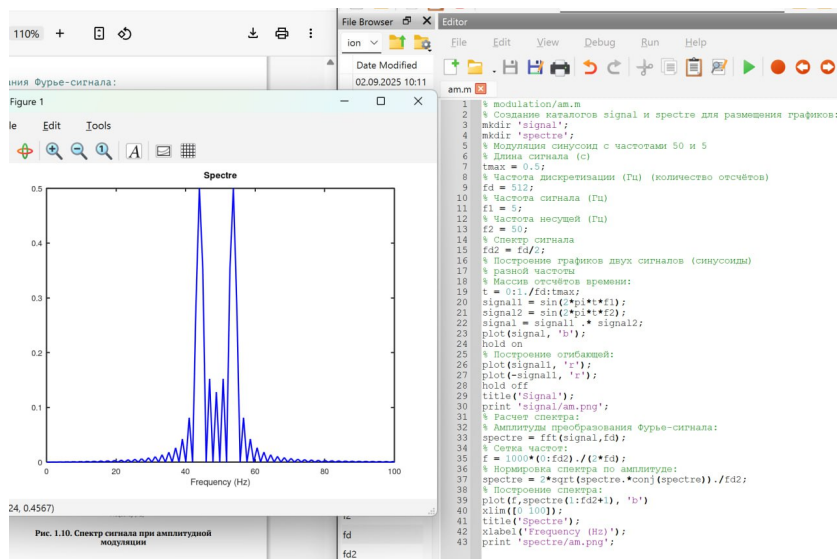


Рис. 2.16: Спектр сигнала при амплитудной модуляции

Теперь необходимо получить кодированные сигналы для нескольких кодов, проверить свойства самосинхронизуемости кодов по заданной битовой последовательности.

Создаем для работы каталог coding и в нём файлы main.m, maptowave.m, unipolar.m, ami.m, bipolar, diffmanс.m, calcspectre.m(рис. 2.17)

Имя	Дата изменения	Тип
main.m	02.09.2025 10:15	Файл "М"
maptowave.m	02.09.2025 10:16	Файл "М"
unipolar.m	02.09.2025 10:18	Файл "М"
ami.m	02.09.2025 10:19	Файл "М"
bipolarnrz.m	02.09.2025 10:19	Файл "М"
bipolarrrz.m	02.09.2025 10:20	Файл "М"
manchester.m	02.09.2025 10:20	Файл "М"
diffmanс.m	02.09.2025 10:21	Файл "М"
calcspectre.m	02.09.2025 10:21	Файл "М"

Рис. 2.17: Подготовка рабочего пространства

Затем убеждаемся, что у нас установлен пакет signal, последовательно добавляем необходимый код в файлы

В файле main.m подключаем пакет signal и задаем 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 \hookrightarrow свойства самосинхронизации: data_sync=[0 0 0 0 0 0 0 1 1 1 1 1 1]; % Входная кодовая последовательность для построения \hookrightarrow спектра сигнала: data_spectre=[0 1 0 1 0 1 0 1 0 1 0 1]; % Создание каталогов signal, sync и spectre для \hookrightarrow размещения графиков: mkdir 'signal'; mkdir 'sync'; mkdir 'spectre'; axis("auto");

Затем в этом же файле пропишем вызовы функций для построения графиков модуляций кодированных сигналов для кодовой последовательности data: % 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';

Затем в этом же файле пропишем вызовы функций для построения графиков модуляций кодированных сигналов для кодовой последовательности data_sync: % 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'; Далее
в этом же файле пропишем вызовы функций для построения графи-
ков спектров: % Униполярное кодирование: wave=unipolar(data_spectre);
spectre=calcspectre(wave); title('Unipolar'); print 'spectre/unipolar.png'; % Коди-
рование 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 манче-
стерское кодирование: wave=diffmanc(data_spectre); spectre=calcspectre(wave);
title('Differential Manchester'); print 'spectre/diffmanc.png';
```

В остальных файлах прописываем функции построения графиков(рис. 2.18) и (рис. 2.19).

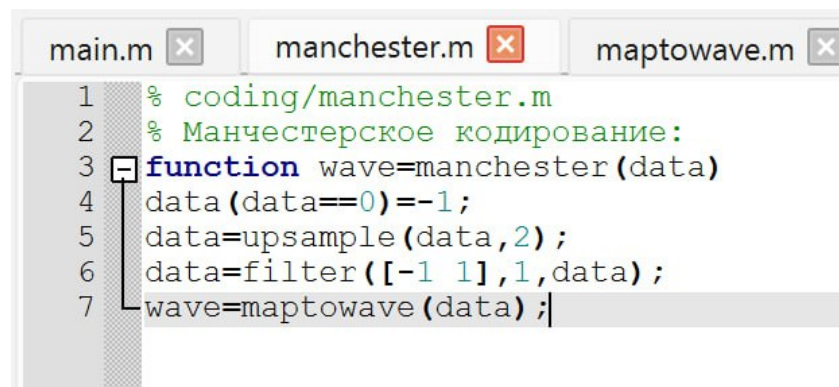


Рис. 2.18: Функции

```

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

```

Рис. 2.19: Функции

Запускаем главный сценарий и получаем следующие результаты(рис. 2.20), (рис. 2.21) и (рис. 2.22).

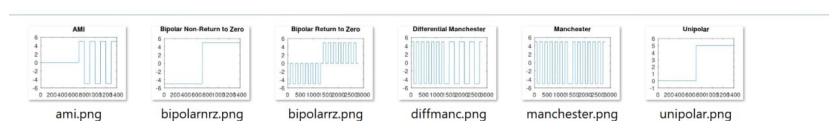


Рис. 2.20: Графики



Рис. 2.21: Графики



Рис. 2.22: Графики

3 Выводы

В ходе работы было произведено знакомство с Octave. Были также изучены методы кодирования и модуляции сигнала, определены спектры и параметры сигнала, продемонстрированы принципы модуляции сигнала на примере аналоговой амплитудной модуляции и исследованы свойства самосинхронизации сигнала

Список литературы

ТУИС