Отчёт по лабораторной работе №1

Дисциплина: Сетевые технологии

Ибрахим Мохсейн Алькамаль

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной работы является изучение методов кодирования и модуляции сигналов с помощью высокоуровнего 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

2.1 Построение графиков в Octave

Запускаем Octave, создаем новый сценарий под названием plot_sin.m. В окне редактора повторяем листинг по построению графика функции y = sin x + 1/3 sin 3x + 1/5 sin 5x на интервале [−10; 10] (рис. [**fig:001**]).

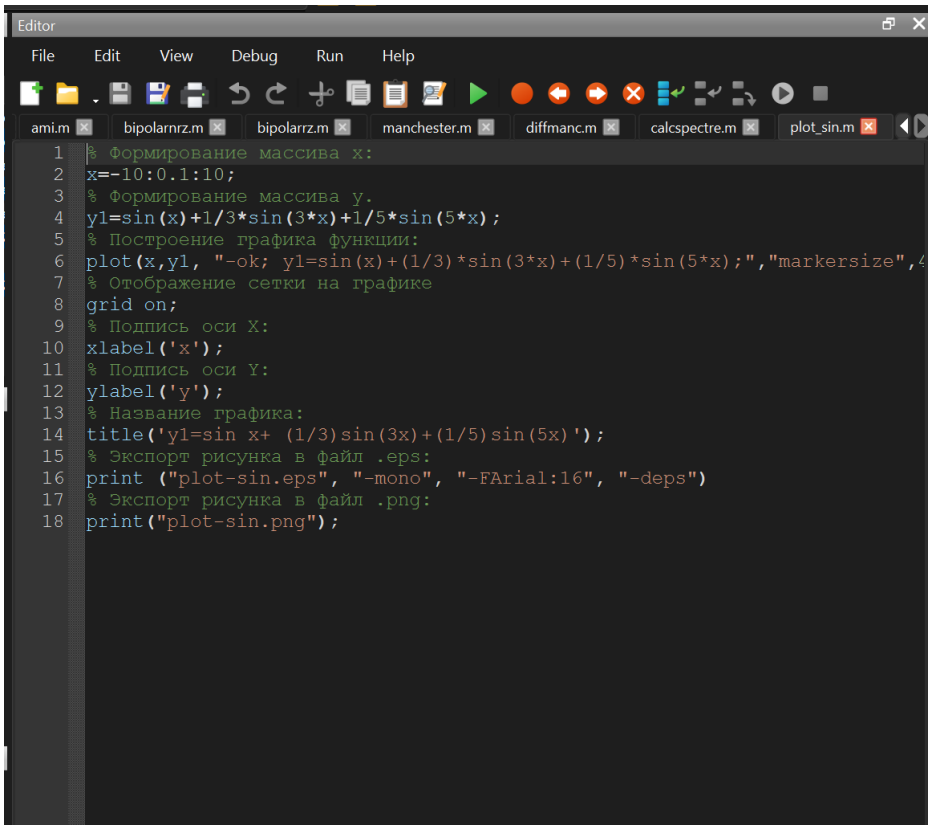


Рисунок 2.1: Листинг файла plot_sin.m

Запускаем сценарий на выполнение, открывается окно с графиком (рис. [**fig:002**]). В рабочем каталоге появляются файла с графиками в форматах .eps, .png (рис. [**fig:003**]).

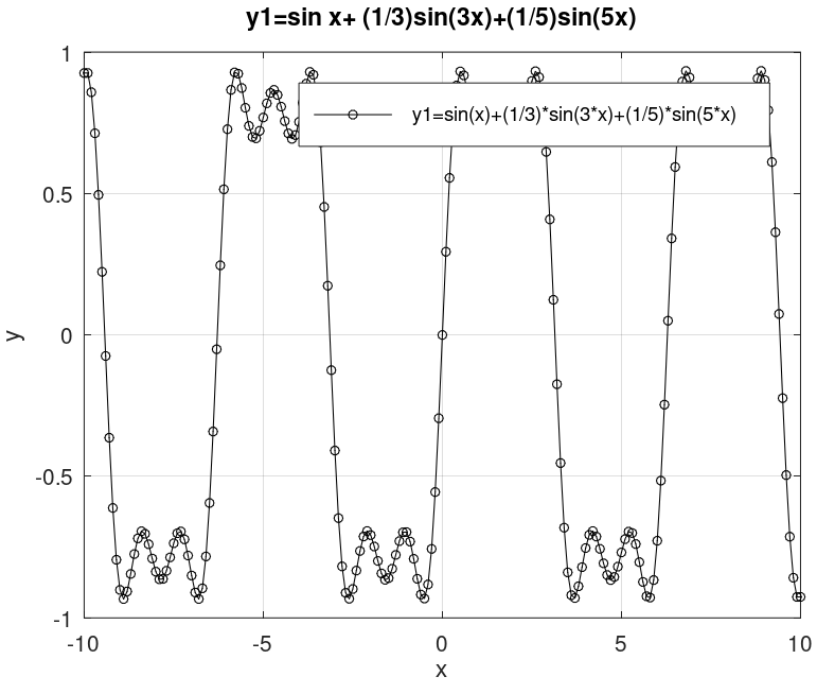


Рисунок 2.2: График функций y1 на интервале −10; 10

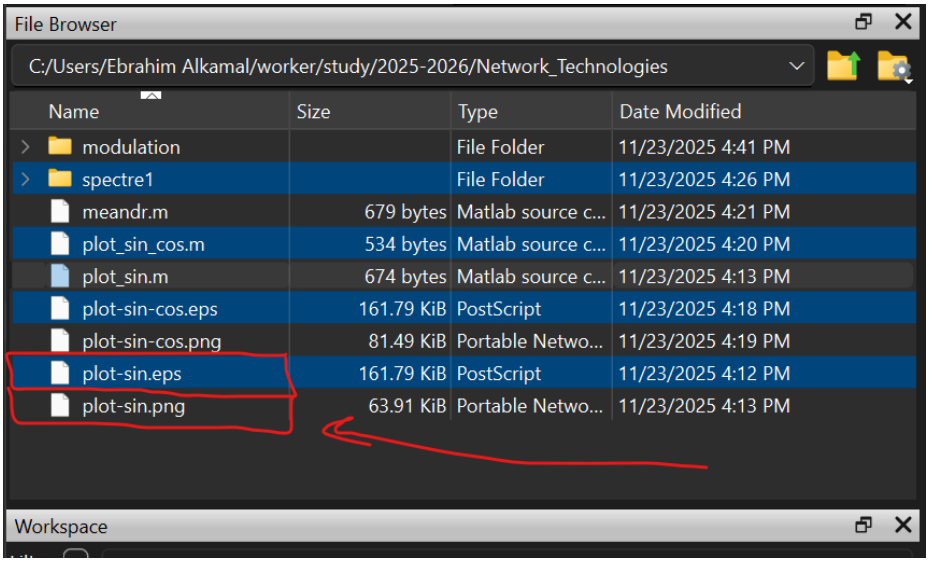


Рисунок 2.3: Файлы .eps, .png

Сохраним сценарий под названием plot_sin_cos.m и изменим его так, чтобы наодном графике располагались отличающиеся по типу линий графики функций y1 = sin x + 1/3 sin 3x + 1/5 sin 5x, y2 = cos x + 1/3 cos 3x + 1/5 cos 5x. Итоговый

листинг (рис. [**fig:004**]).

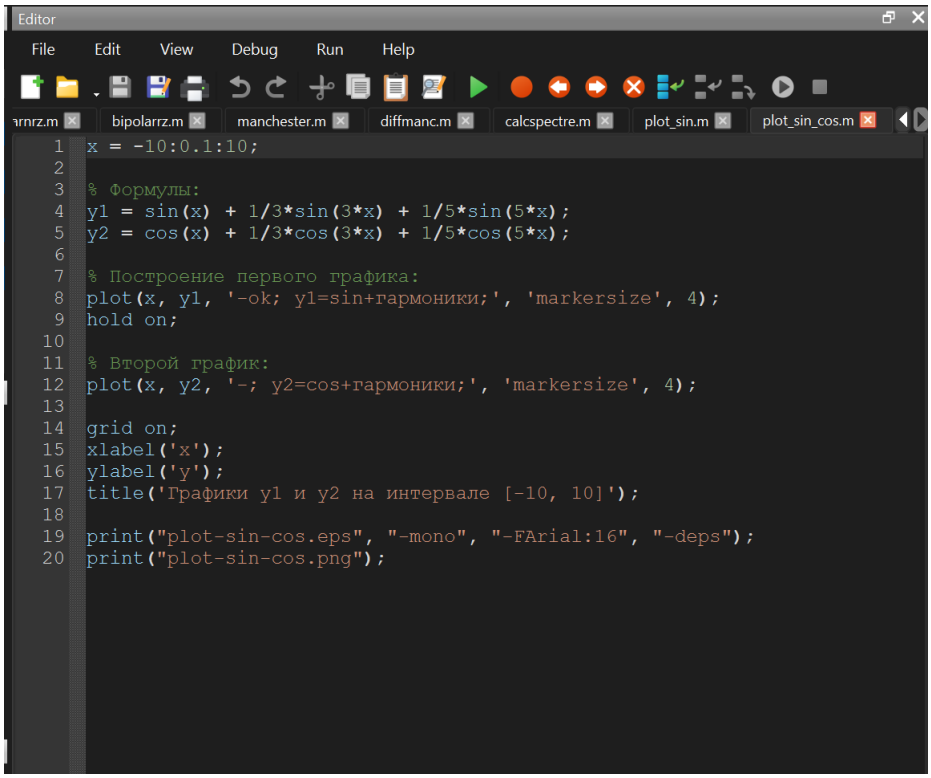


Рисунок 2.4: Листинг файла plot_sin_cos.m

Запускаем, получаем еще один график (рис. [**fig:005**]).

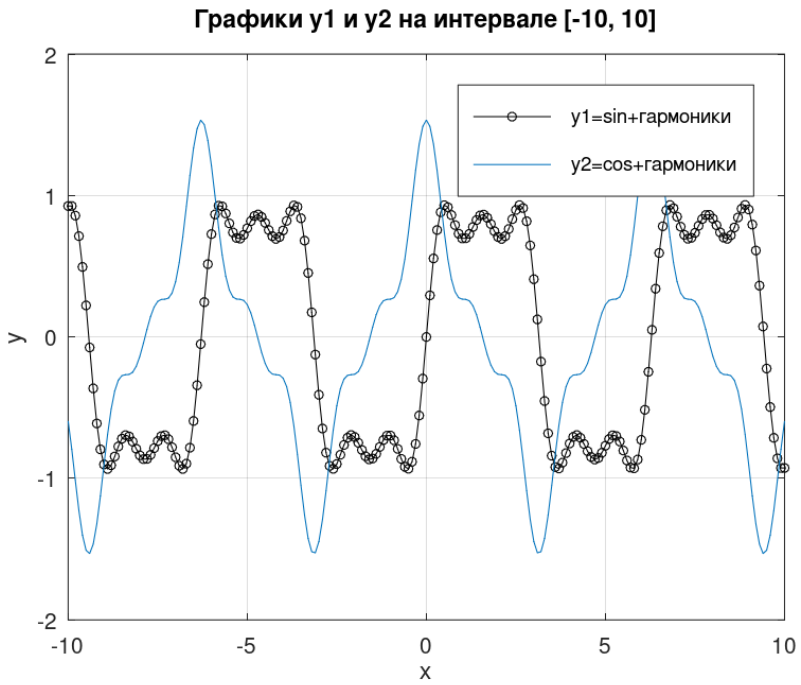


Рисунок 2.5: График функций y1 и y2 на интервале −10; 10

2.2 Разложение импульсного сигнала в частичный ряд Фурье

Создадим новый сценарий meandr.m. В коде зададим начальные значения. Вычислим амплитуду гармоник и заполним массивы гармоник и элементов ряда. Далее задаём массив значений гармоник массив элементов ряда. 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. Также экспортируем полученный график в файл в формате .png (рис. [**fig:006**]), (рис. [**fig:007**]).

```
% meandr.m
% количество отсчетов (гармоник):
N=8;
% частота дискретизации:
t=-1:0.01:1;
% значение амплитуды:
A=1;
% период:
T=1;
% 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);
% массив гармоник:
harmonics=cos(2 * pi * nh' * t/T);
% массив элементов ряда:
s1=harmonics.*repmat(Am',1,length(t));

% Суммирование ряда:
s2=cumsum(s1);
% Построение графиков:
for k=1:N
subplot(4,2,k)
plot(t, s2(k,:))
end

print("plot-meandr-cos.png");
```

Рисунок 2.6: Листинг файла meandr.m

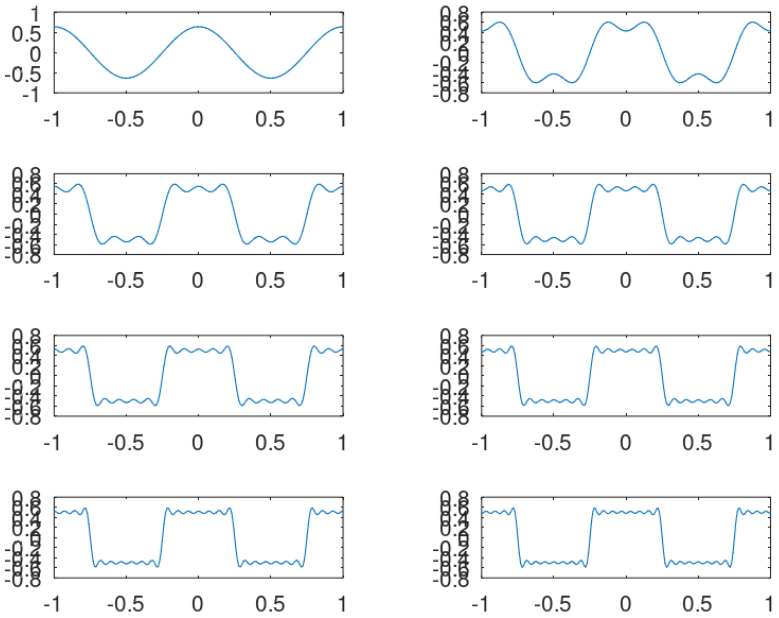


Рисунок 2.7: Меандр через косинусы

Также реализуем меандр через синусы (рис. [**fig:008**]), (рис. [**fig:009**]).

```
% meandr.m
% количество отсчетов (гармоник):
N=8;
% частота дискретизации:
t=-1:0.01:1;
% значение амплитуды:
A=1;
% период:
T=1;
% 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);
% массив гармоник:
harmonics=sin(2 * pi * nh' * t/T);
% массив элементов ряда:
s1=harmonics.*repmat(Am',1,length(t));

% Суммирование ряда:
s2=cumsum(s1);
% Построение графиков:
for k=1:N
  subplot(4,2,k)
  plot(t, s2(k,:))
end

print("plot-meandr-cos.png");
```

Рисунок 2.8: Листинг файла meandr.m

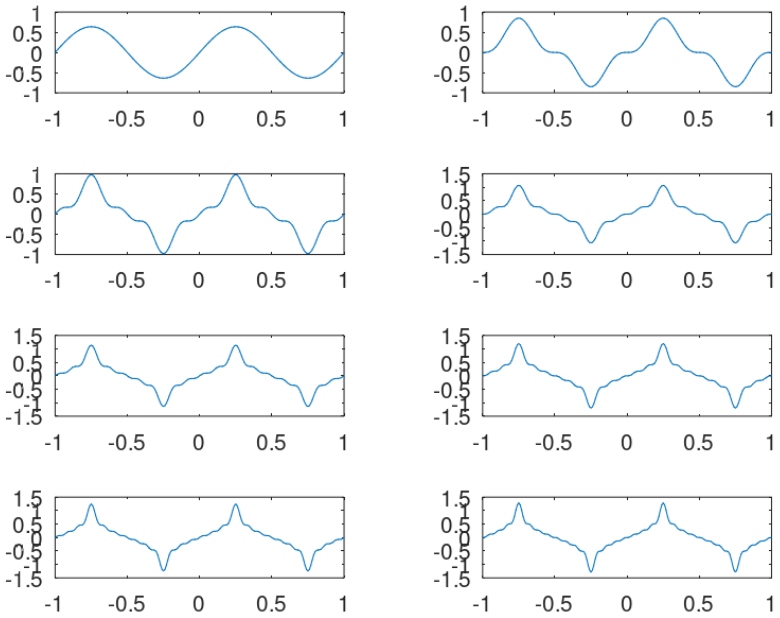


Рисунок 2.9: Меандр через синусы

2.3 Определение спектра и параметров сигнала

Создадим в рабочем каталоге каталог spectre1 и в нем новый сценарий spectre.m. В коде сценария зададим начальные значения, а также два синусоидальных сигнала разной частоты, построим графики сигналов (рис. [**fig:010**]), (рис. [**fig:011**]).

```m
% spectre1/spectre.m
% Создание каталогов signal и spectre для размещения графиков:
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;
% Массив отсчётов времени:
t = 0:1./fd:tmax;
% Спектр сигнала:
fd2 = fd/2;
% Два сигнала разной частоты:
signal1 = a1*sin(2*pi*t*f1);
signal2 = a2*sin(2*pi*t*f2);

% График 1-го сигнала:
plot(signal1,'b');
% График 2-го сигнала:
hold on
plot(signal2,'r');
hold off
title('Signal');
% Экспорт графика в файл в каталоге signal:
print 'signal/spectre.png';
```

Рисунок 2.10: Листинг файла spectre.m

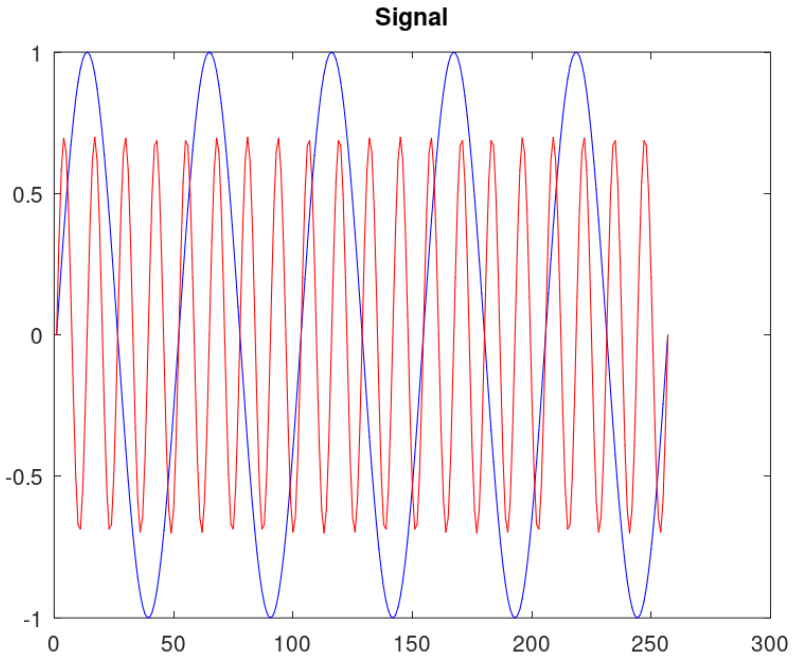


Рисунок 2.11: Графики сигналов разной частоты

Затем с помощью быстрого преобразования Фурье найдем спектры сигналов, добавив в файл spectre.m код из мануала в ТУИСе. Учитывая реализацию преобразования Фурье, скорректируем график спектра (рис. [**fig:012**]): отбросим дублирующие отрицательные частоты, а также примим в расчёт то, что на каждом шаге вычисления быстрого преобразования Фурье происходит суммирование амплитуд сигналов.

```
% Посчитаем спектр
% Амплитуды преобразования Фурье сигнала 1:
spectre1 = abs(fft(signal1,fd));
% Амплитуды преобразования Фурье сигнала 2:
spectre2 = abs(fft(signal2,fd));
% Построение графиков спектров сигналов:
plot(spectre1,'b');
hold on
plot(spectre2,'r');
hold off
title('Spectre');
print 'spectre/spectre.png';

% Исправление графика спектра
% Сетка частот:
f = 1000*(0:fd2)./(2*fd);
% Нормировка спектров по амплитуде:
spectre1 = 2*spectre1/fd2;
spectre2 = 2*spectre2/fd2;
% Построение графиков спектров сигналов:
plot(f,spectre1(1:fd2+1),'b');
hold on

plot(f,spectre2(1:fd2+1),'r');
hold off
xlim([0 100]);
title('Fixed spectre');
xlabel('Frequency (Hz)');
print 'spectre/spectre_fix.png';

```

Рисунок 2.12: Листинг файла spectre.m

Получим следующие графики: график спектров синусоидальных сигналов (рис. [**fig:013**]) и исправленный график спектров синусоидальных сигналов (рис. [**fig:014**]).

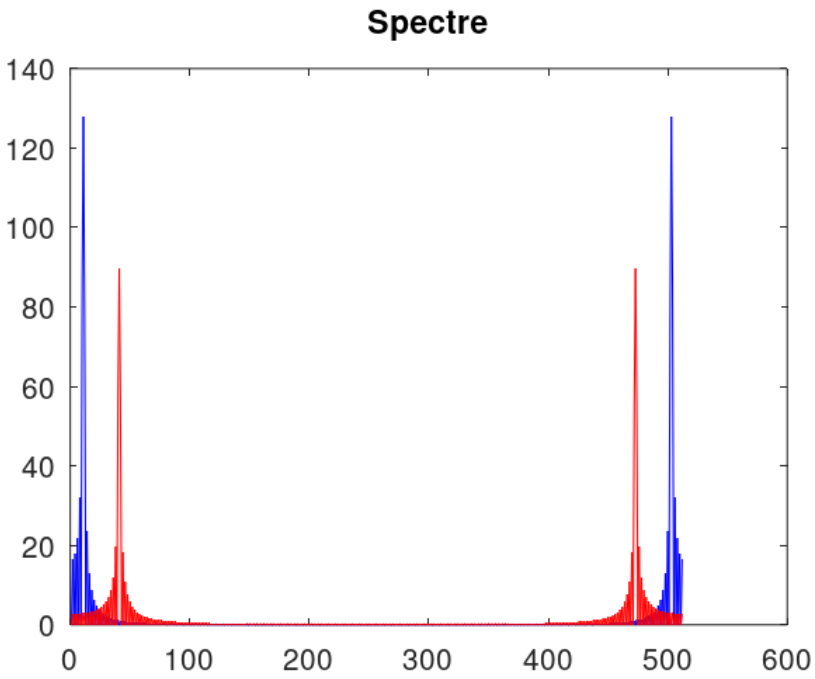


Рисунок 2.13: График спектра синусоидальных сигналов

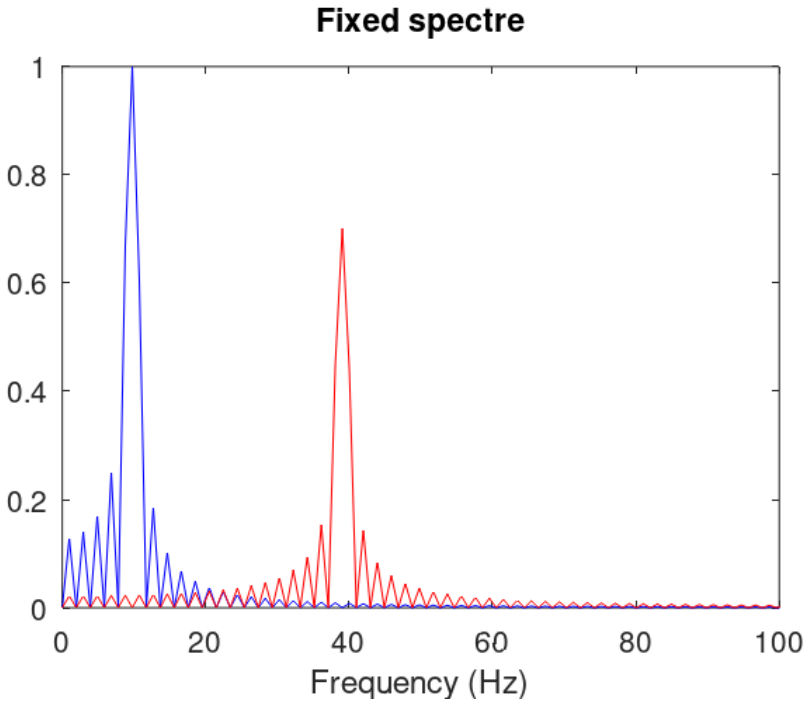


Рисунок 2.14: Исправленный график спектров синусоидальных сигналов

Найдем спектр суммы рассмотренных сигналов, создадим каталог spectr_sum и в нем spectre_sum.m (рис. [**fig:015**]), (рис. [**fig:016**]).

spectre_sum.m

```
% spectr_sum/spectre_sum.m
% Создание каталогов signal и spectre для размещения графиков:
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:X
% 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';

```

Рисунок 2.15: Листинг файла spectre_sum.m

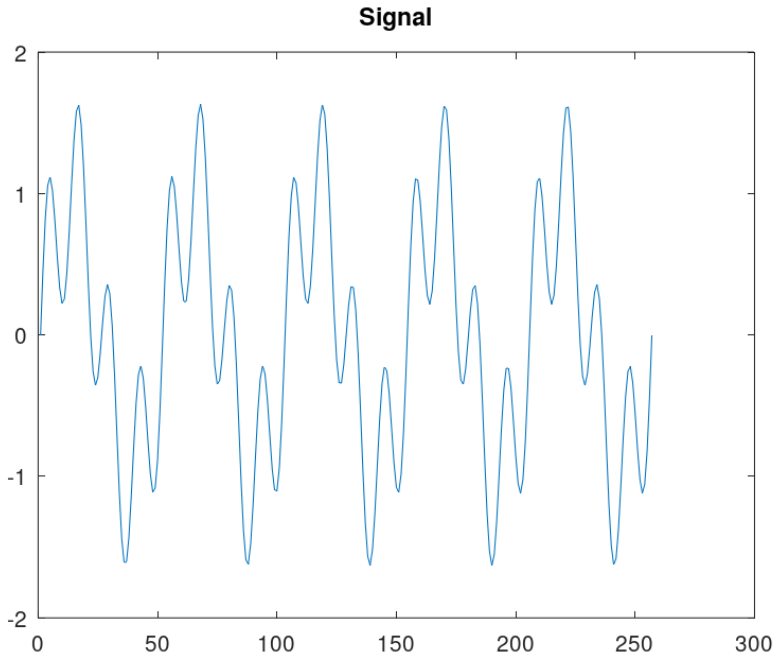


Рисунок 2.16: Суммарный сигнал

В результате должен получится аналогичный предыдущему результат (рис. [**fig:017**]), т.е. спектр суммы сигналов должен быть равен сумме спектров сигналов, что вытекает из свойств преобразования Фурье.

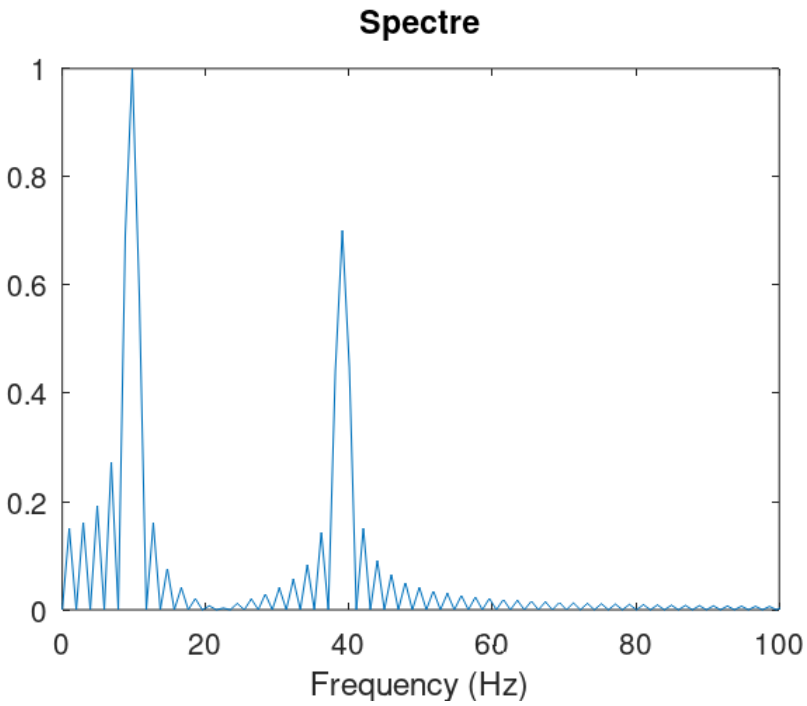


Рисунок 2.17: Спектр суммарного сигнала

2.4 Амплитудная модуляция

В рабочем каталоге создадим каталог modulation и в нём новый сценарий с именем am.m. Добавим в него код из мануала (рис. [**fig:018**]).

am.m

```
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 offZ
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';
```

Рисунок 2.18: Листинг файла am.m

В результате получаем, что спектр произведения представляет собой свертку спектров (рис. [**fig:019**]), (рис. [**fig:020**]).

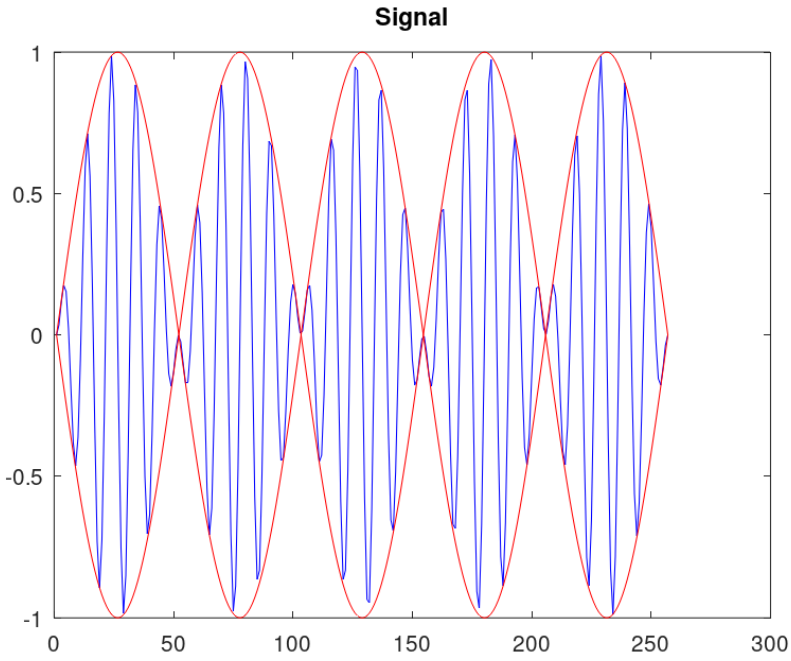


Рисунок 2.19: Сигнал и огибающая при амплитудной модуляции

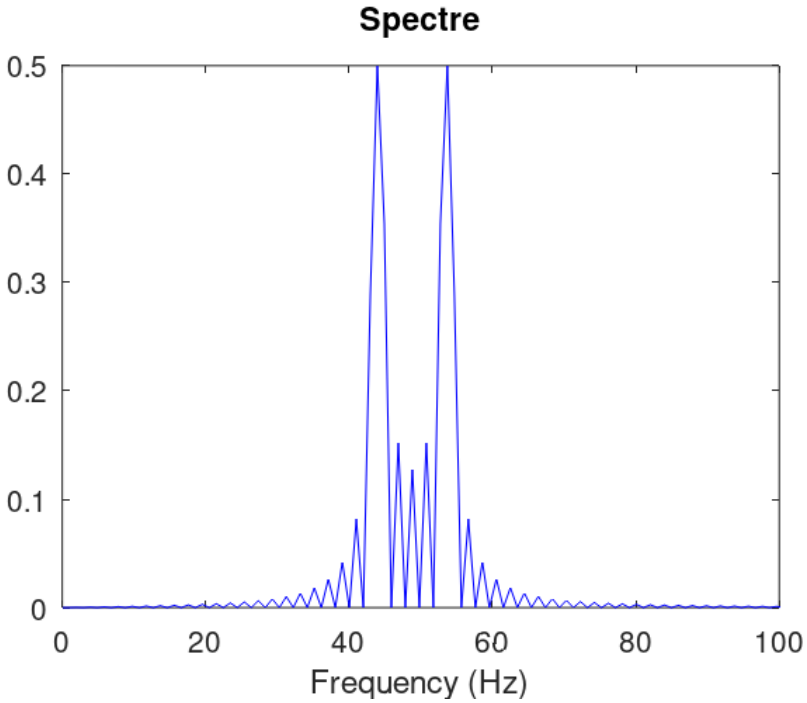


Рисунок 2.20: Спектр сигнала при амплитудной модуляции

2.5 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

В рабочем каталоге создадим каталог coding и в нём файлы main.m, maptowave.m, unipolar.m, ami.m, bipolarnrz.m, bipolarrz.m, manchester.m, diffmanc.m, calcspectre.m.

В окне интерпретатора команд проверяем, установлен ли пакет расширений signal: pkg list. Так как он не установлен, то устанавливаем его: pkg list -forge и pkg install control signal (рис. [**fig:021**]).

```
>> pkg list
Package Name         | Version | Installation directory
---------------------+---------+-----------------------
               audio |   2.0.9 | ...\Octave-10.3.0\mingw64\share\octave\packages\audio-2.0.9
              biosig |   3.9.0 | ...\Octave-10.3.0\mingw64\share\octave\packages\biosig-3.9.0
             cfitsio |   0.0.7 | ...\Octave-10.3.0\mingw64\share\octave\packages\cfitsio-0.0.7
               coder |  1.10.1 | ...\Octave-10.3.0\mingw64\share\octave\packages\coder-1.10.1
      communications |   1.2.7 | ...\mingw64\share\octave\packages\communications-1.2.7
             control |   4.1.3 | ...\Octave-10.3.0\mingw64\share\octave\packages\control-4.1.3
      data-smoothing |   1.3.0 | ...\mingw64\share\octave\packages\data-smoothing-1.3.0
            database |   2.4.4 | ...\Octave-10.3.0\mingw64\share\octave\packages\database-2.4.4
           dataframe |   1.2.0 | ...\mingw64\share\octave\packages\dataframe-1.2.0
               dicom |   0.6.1 | ...\Octave-10.3.0\mingw64\share\octave\packages\dicom-0.6.1
           financial |   0.5.4 | ...\mingw64\share\octave\packages\financial-0.5.4
             fl-core |   1.0.2 | ...\Octave-10.3.0\mingw64\share\octave\packages\fl-core-1.0.2
fuzzy-logic-toolkit |   0.6.2 | ...\mingw64\share\octave\packages\fuzzy-logic-toolkit-0.6.2
                 ga |  0.10.4 | ...\Octave-10.3.0\mingw64\share\octave\packages\ga-0.10.4
            general |   2.1.3 | ...\Octave-10.3.0\mingw64\share\octave\packages\general-2.1.3
      generate_html |   0.3.3 | ...\mingw64\share\octave\packages\generate_html-0.3.3
           geometry |   4.1.0 | ...\Octave-10.3.0\mingw64\share\octave\packages\geometry-4.1.0
                gsl |   2.1.1 | ...\Octave-10.3.0\mingw64\share\octave\packages\gsl-2.1.1
              image |  2.18.1 | ...\Octave-10.3.0\mingw64\share\octave\packages\image-2.18.1
  image-acquisition |   0.3.3 | ...\mingw64\share\octave\packages\image-acquisition-0.3.3
 instrument-control |   0.9.5 | ...\mingw64\share\octave\packages\instrument-control-0.9.5
           interval |   3.2.1 | ...\Octave-10.3.0\mingw64\share\octave\packages\interval-3.2.1
                 io |   2.7.0 | ...\Octave-10.3.0\mingw64\share\octave\packages\io-2.7.0
     linear-algebra |   2.2.3 | ...\mingw64\share\octave\packages\linear-algebra-2.2.3
               lssa |   0.1.4 | ...\Octave-10.3.0\mingw64\share\octave\packages\lssa-0.1.4
              ltfat |   2.6.0 | ...\Octave-10.3.0\mingw64\share\octave\packages\ltfat-2.6.0
            mapping |   1.4.3 | ...\Octave-10.3.0\mingw64\share\octave\packages\mapping-1.4.3
            matgeom |   1.2.4 | ...\Octave-10.3.0\mingw64\share\octave\packages\matgeom-1.2.4
      miscellaneous |   1.3.1 | ...\mingw64\share\octave\packages\miscellaneous-1.3.1
               mqtt |   0.0.5 | ...\Octave-10.3.0\mingw64\share\octave\packages\mqtt-0.0.5
                nan |   3.7.0 | ...\Octave-10.3.0\mingw64\share\octave\packages\nan-3.7.0
              netcdf |  1.0.18 | ...\Octave-10.3.0\mingw64\share\octave\packages\netcdf-1.0.18
              nurbs |   1.4.4 | ...\Octave-10.3.0\mingw64\share\octave\packages\nurbs-1.4.4
                ocs |   0.1.5 | ...\Octave-10.3.0\mingw64\share\octave\packages\ocs-0.1.5
            octproj |   3.1.0 | ...\Octave-10.3.0\mingw64\share\octave\packages\octproj-3.1.0
              optim |   1.6.2 | ...\Octave-10.3.0\mingw64\share\octave\packages\optim-1.6.2
       optiminterp |   0.3.7 | ...\mingw64\share\octave\packages\optiminterp-0.3.7
          parallel |   4.0.2 | ...\Octave-10.3.0\mingw64\share\octave\packages\parallel-4.0.2
        quaternion |   2.4.0 | ...\mingw64\share\octave\packages\quaternion-2.4.0
          queueing |   1.2.8 | ...\Octave-10.3.0\mingw64\share\octave\packages\queueing-1.2.8
            signal |   1.4.6 | ...\Octave-10.3.0\mingw64\share\octave\packages\signal-1.4.6
```

Рисунок 2.21: Проверка правильности установки пакета signal

В файле main.m подключаем пакет signal и задаем входные кодовые последовательности (рис. [**fig:022**]).

main.m

```
% 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 самосинхрон
data_sync=[0 0 0 0 0 0 0 1 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 0 1];
% Создание каталогов signal, sync и spectre для размещения графиков:
mkdir 'signal';
mkdir 'sync';
mkdir 'spectre';
axis("auto");
```

Рисунок 2.22: Задаем входные кодовые последовательности

Затем в этом же файле пропишем вызовы функций для построения графиков модуляций кодированных сигналов для кодовой последовательности data (рис. [**fig:023**]).

```
15 % Униполярное кодирование
16 wave=unipolar(data);
17 plot(wave);
18 ylim([-1 6]);
19 title('Unipolar');
20 print 'signal/unipolar.png';
21 % Кодирование ami
22 wave=ami(data);
23 plot(wave)
24 title('AMI');
25 print 'signal/ami.png';
26 % Кодирование NRZ
27 wave=bipolarnrz(data);
28 plot(wave);
29 title('Bipolar Non-Return to Zero');
30 print 'signal/bipolarnrz.png';
31 % Кодирование RZ
32 wave=bipolarrz(data);
33 plot(wave)
34 title('Bipolar Return to Zero');
35 print 'signal/bipolarrz.png';
36 % Манчестерское кодирование
37 wave=manchester(data);
38 plot(wave)
39 title('Manchester');
40 print 'signal/manchester.png';
41
42 % Дифференциальное манчестерское кодирование
43 wave=diffmanc(data);
44 plot(wave)
45 title('Differential Manchester');
46 print 'signal/diffmanc.png';
```

Рисунок 2.23: Вызовы функций для посторения модуляций кодированных сигналов кодовой последовательности data

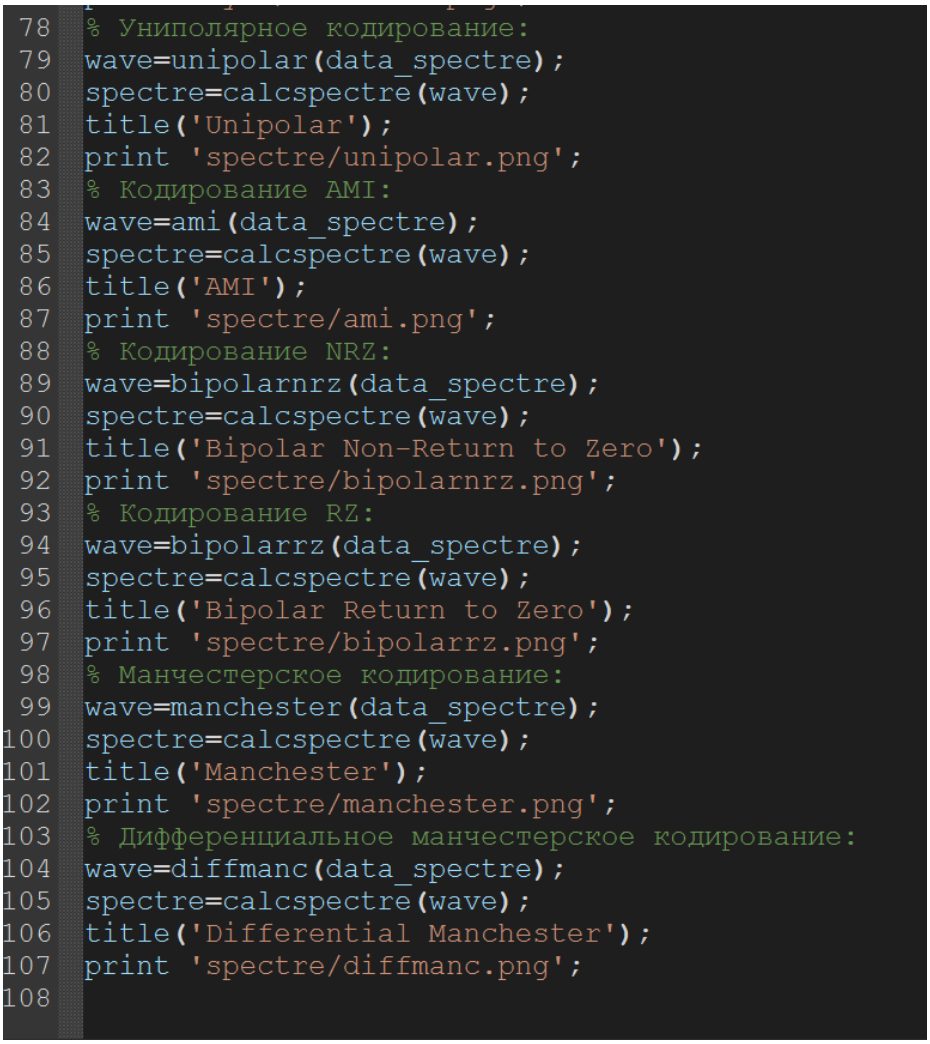
Пропишем вызовы функций для построения графиков модуляций кодированных сигналов для кодовой последовательности data_sync (рис. [**fig:024**]).

main.m

```
47  % Униполярное кодирование
48  wave=unipolar(data_sync);
49  plot(wave);
50  ylim([-1 6]);
51  title('Unipolar');
52  print 'sync/unipolar.png';
53  % Кодирование AMI
54  wave=ami(data_sync);
55  plot(wave)
56  title('AMI');
57  print 'sync/ami.png';
58  % Кодирование NRZ
59  wave=bipolarnrz(data_sync);
60  plot(wave);
61  title('Bipolar Non-Return to Zero');
62  print 'sync/bipolarnrz.png';
63  % Кодирование RZ
64  wave=bipolarrz(data_sync);
65  plot(wave)
66  title('Bipolar Return to Zero');
67  print 'sync/bipolarrz.png';
68  % Манчестерское кодирование
69  wave=manchester(data_sync);
70  plot(wave)
71  title('Manchester');
72  print 'sync/manchester.png';
73  % Дифференциальное манчестерское кодирование
74  wave=diffmanc(data_sync);
75  plot(wave)
76  title('Differential Manchester');
77  print 'sync/diffmanc.png';
```

Рисунок 2.24: Вызовы функций для посторения модуляций кодированных сигналов кодовой последовательности data_sync

Далее в этом же файле пропишем вызовы функций для построения графиков спектров (рис. [**fig:025**]).



```
% 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';

```

Рисунок 2.25: Вызовы функций для посторения графиков спектров

В файле maptowave.m пропишем функцию, которая по входному битовому потоку строит график сигнала (рис. [**fig:026**]).

В файлах unipolar.m (рис. [**fig:027**]), ami.m (рис. [**fig:028**]), bipolarnrz.m (рис. [**fig:029**]), bipolarrz.m (рис. [**fig:030**]), manchester.m (рис. [**fig:031**]), diffmanc.m (рис. [**fig:032**]) пропишем 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);
end
```

Рисунок 2.26: Листинг файла unipolar.m

unipolar.m 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);
end
```

Рисунок 2.27: Листинг файла ami.m

unipolar.m ami.m bipolarnrz.m

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

Рисунок 2.28: Листинг файла bipolarnrz.m

bipolarrz.m

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

Рисунок 2.29: Листинг файла bipolarrz.m

bipolarrz.m 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);
end
```

Рисунок 2.30: Листинг файла manchester.m

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);
```

Рисунок 2.31: Листинг файла diffmanc.m

В файле calcspectre.m пропишем функцию построения спектра сигнала (рис. [**fig:033**]).

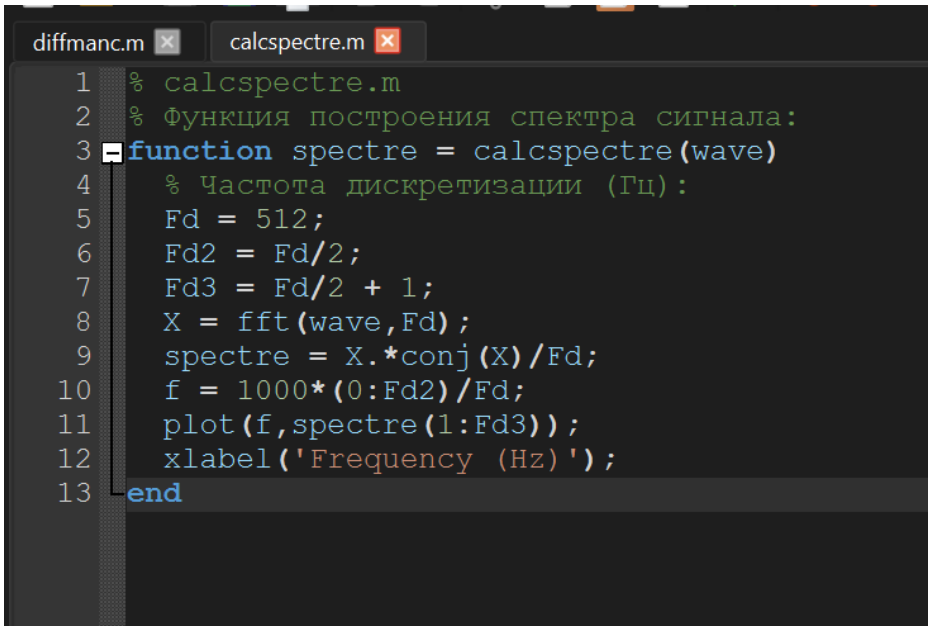


Рисунок 2.32: Листинг файла calcspectre.m

Запустим главный скрипт main.m. В каталоге signal должны быть получены файлы с графиками кодированного сигнала (рис. [**fig:034**]-[**fig:039**]), в каталоге sync — файлы с графиками, иллюстрирующими свойства самосинхронизации (рис. [**fig:040**]-[**fig:045**]), в каталоге spectre — файлы с графиками спектров сигналов (рис. [**fig:046**]-[**fig:051**]).

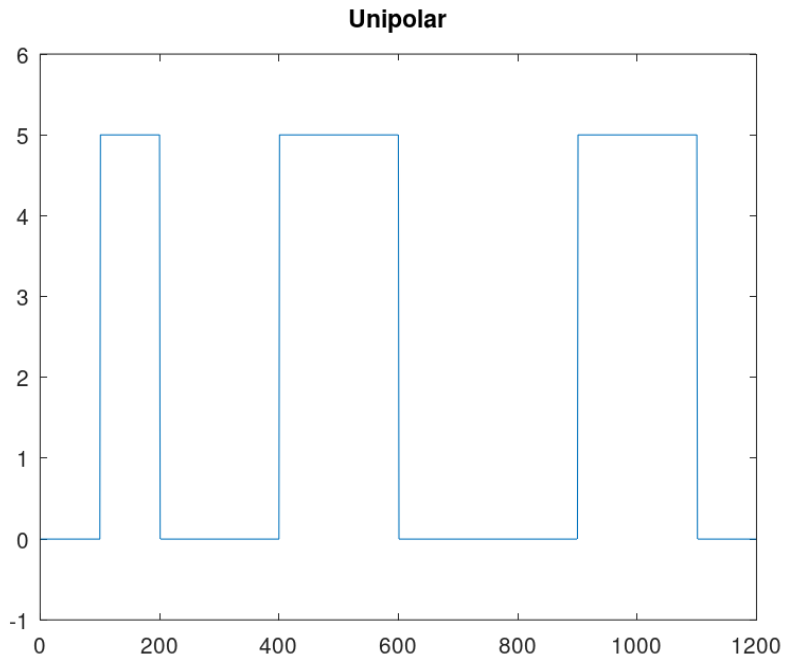


Рисунок 2.33: Униполярное кодирование

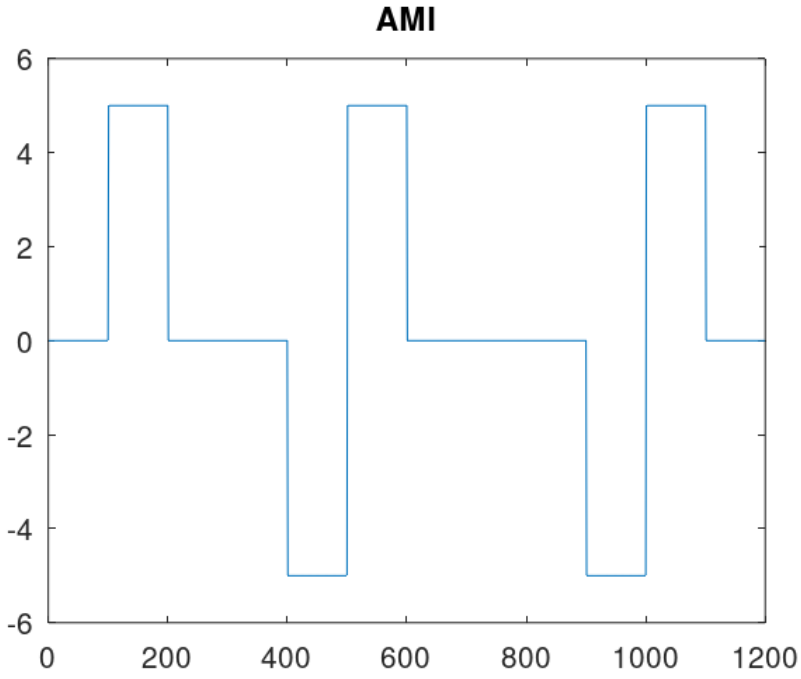


Рисунок 2.34: Кодирование AMI

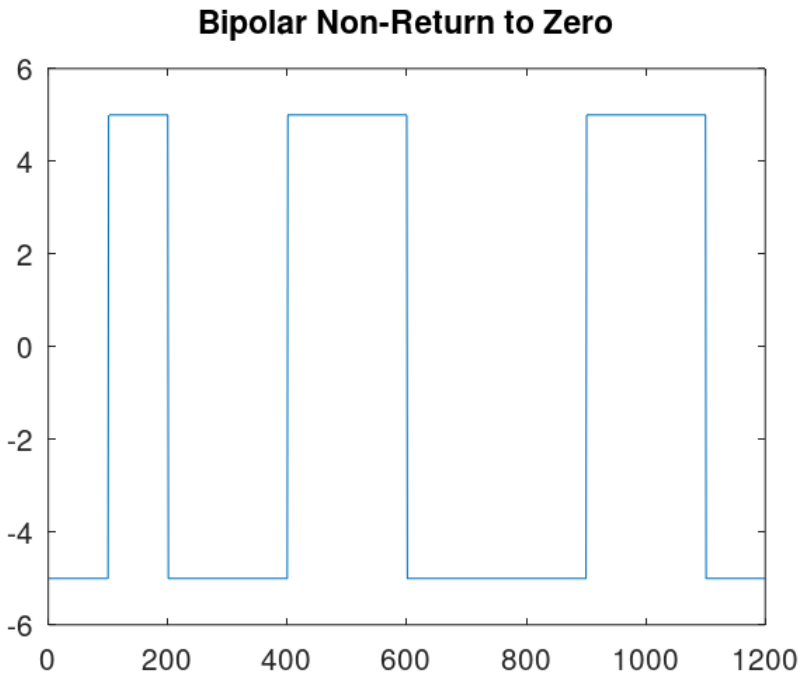


Рисунок 2.35: Кодирование NRZ

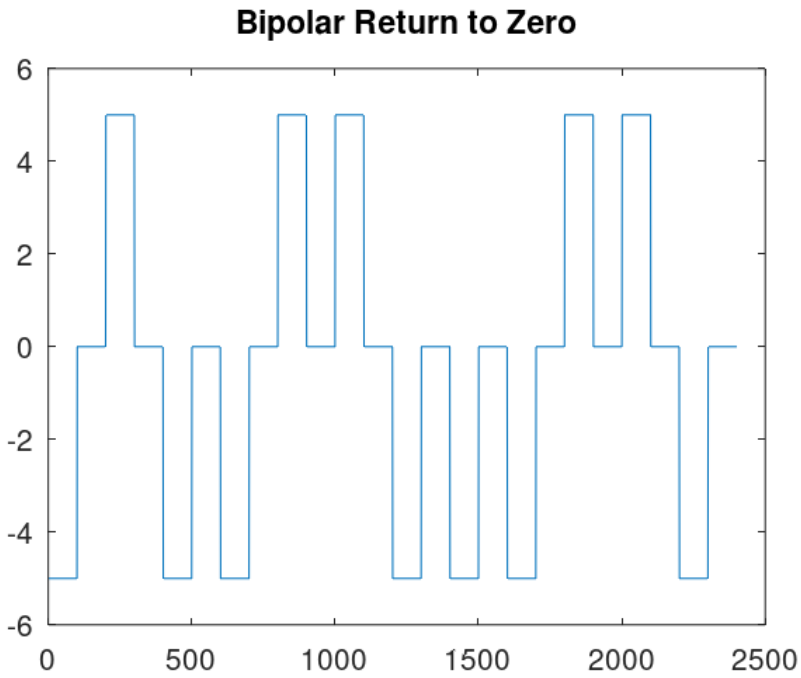


Рисунок 2.36: Кодирование RZ

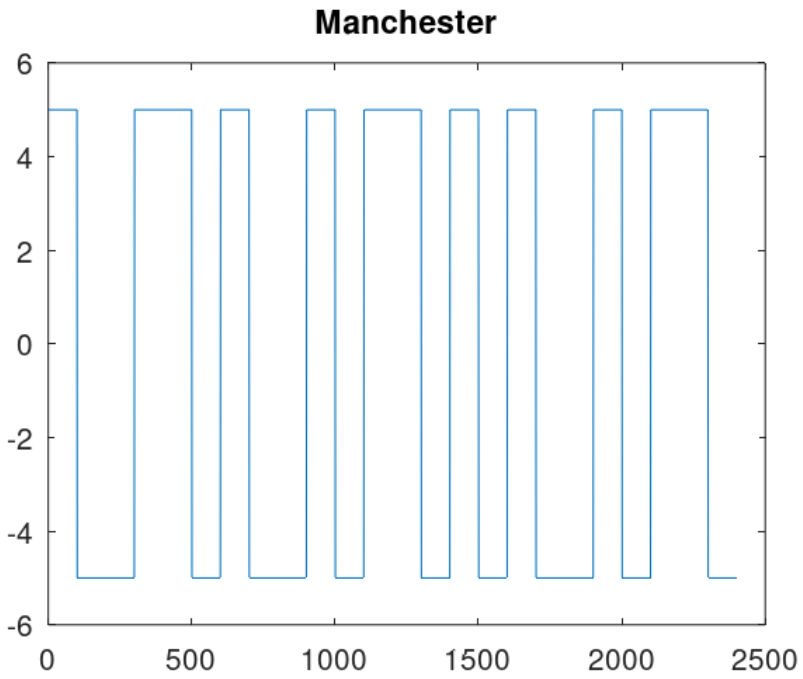


Рисунок 2.37: Манчестерское кодирование

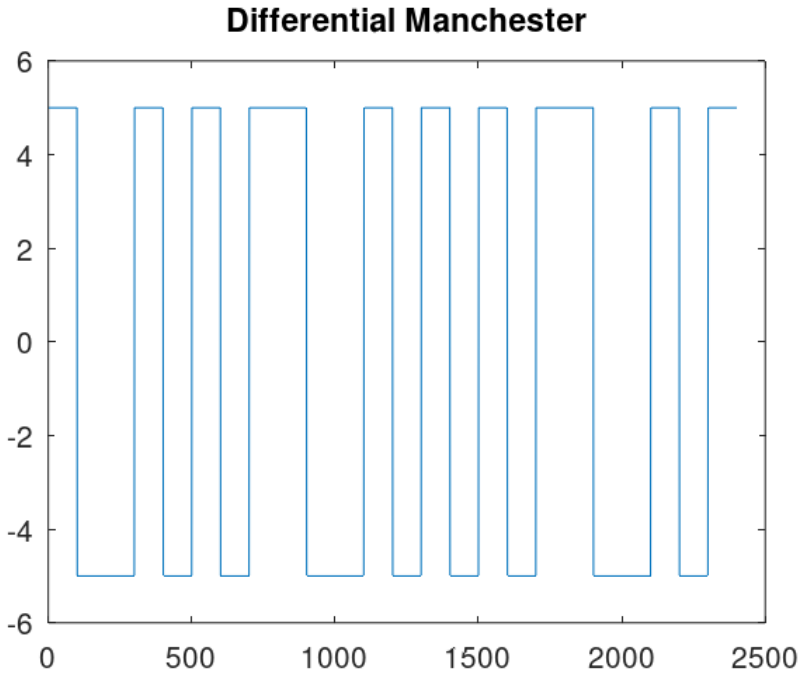


Рисунок 2.38: Дифференциальное манчестерское кодирование

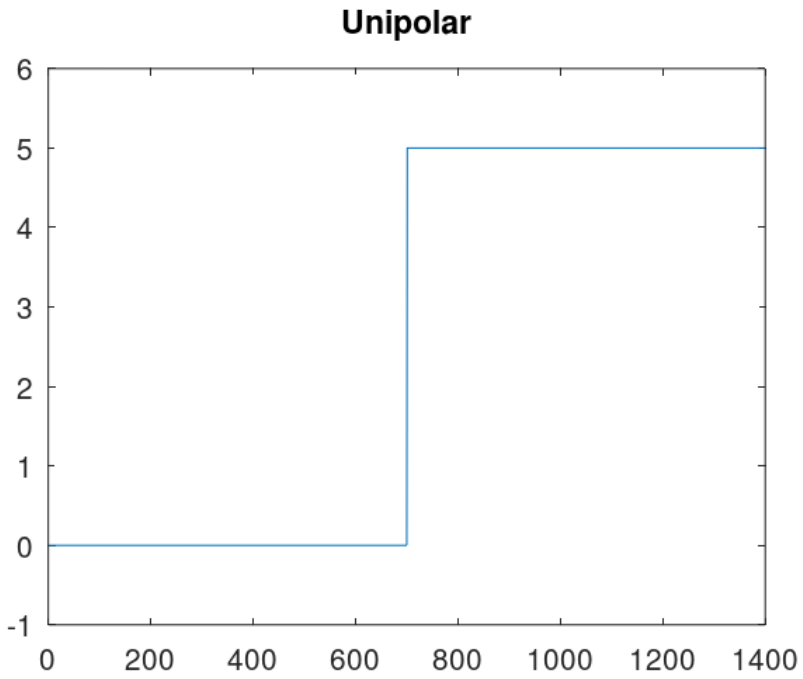


Рисунок 2.39: Униполярное кодирование: нет самосинхронизации

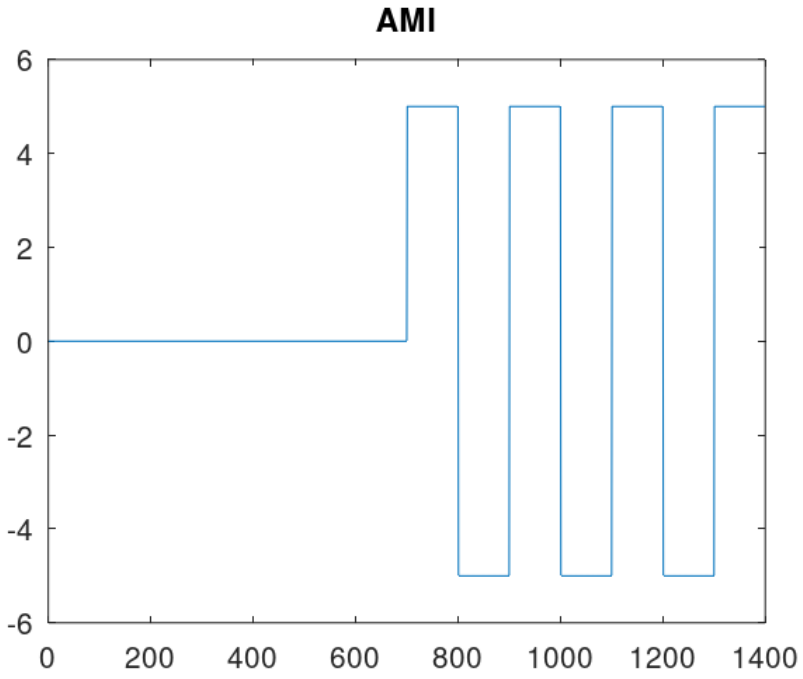


Рисунок 2.40: Кодирование AMI: самосинхронизация при наличии сигнала

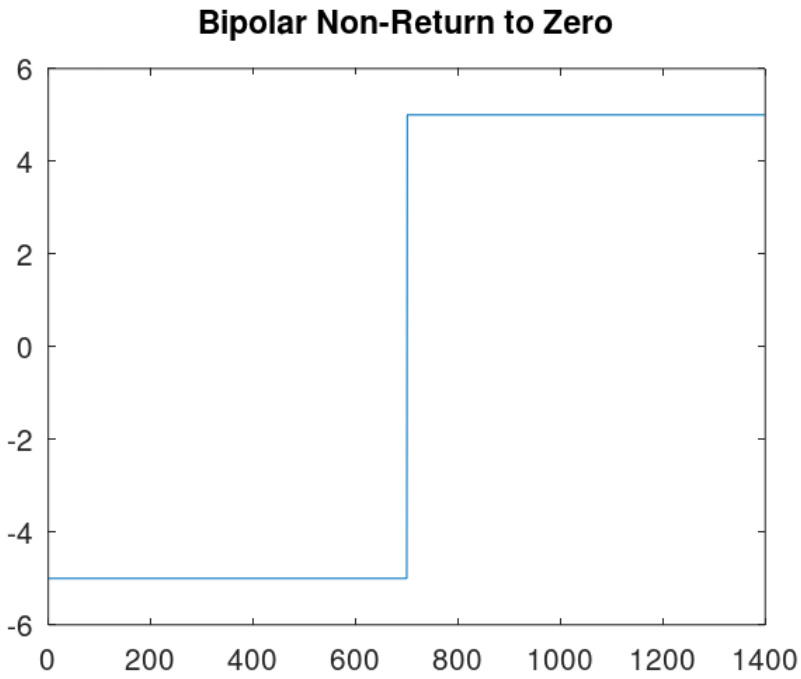


Рисунок 2.41: Кодирование NRZ: нет самосинхронизации

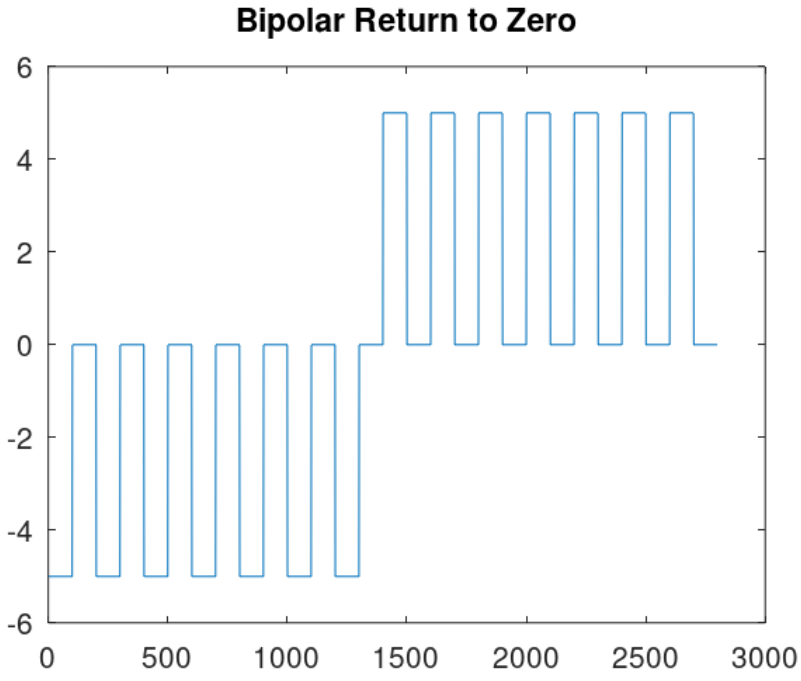


Рисунок 2.42: Кодирование RZ: есть самосинхронизация

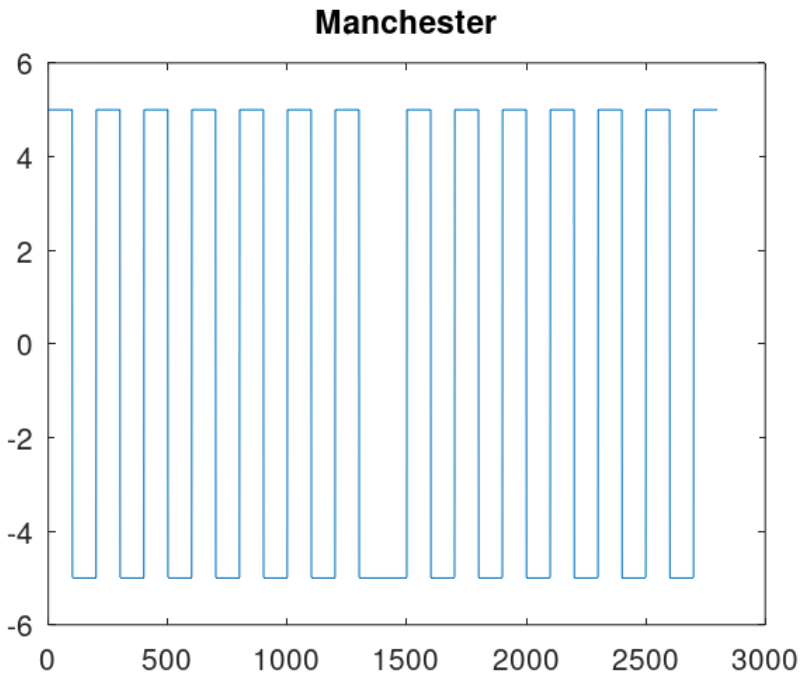


Рисунок 2.43: Манчестерское кодирование: есть самосинхронизация

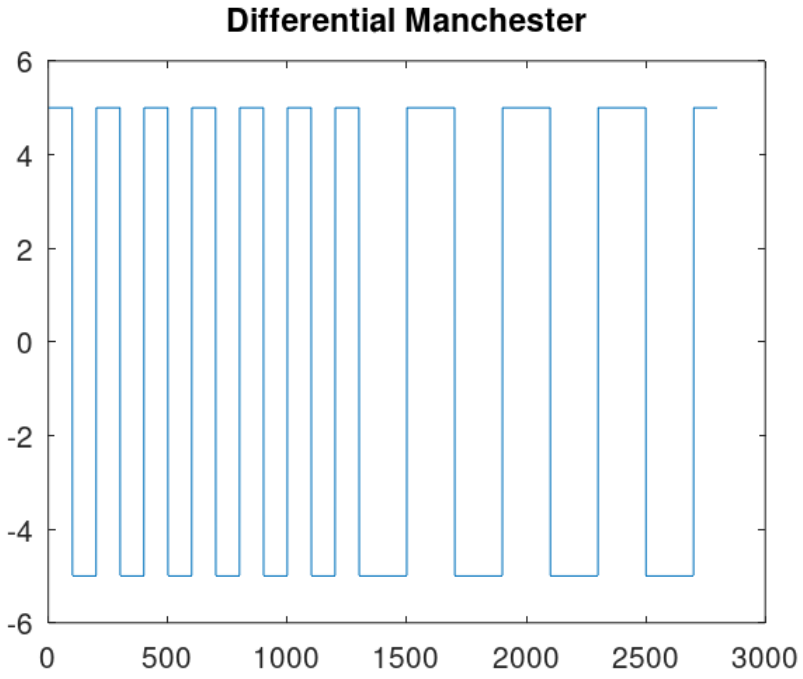


Рисунок 2.44: Дифференциальное манчестерское кодирование: есть самосинхронизация

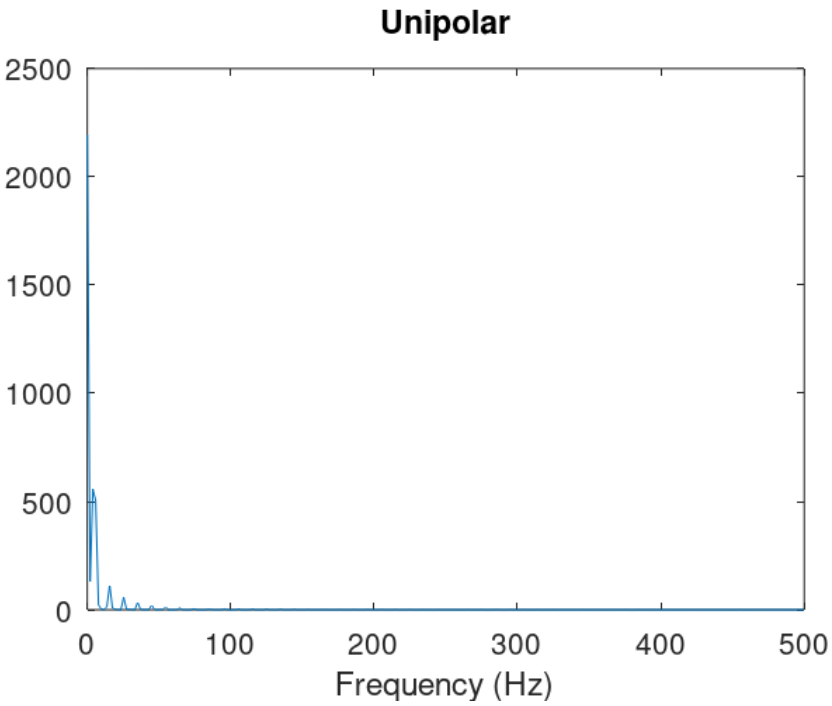


Рисунок 2.45: Униполярное кодирование: спектр сигнала

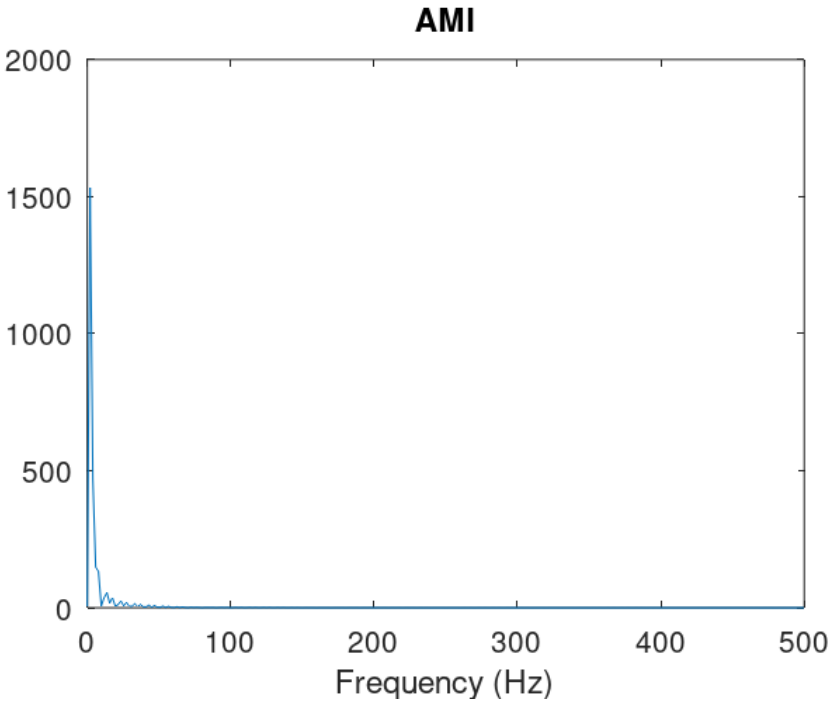


Рисунок 2.46: Кодирование AMI: спектр сигнала

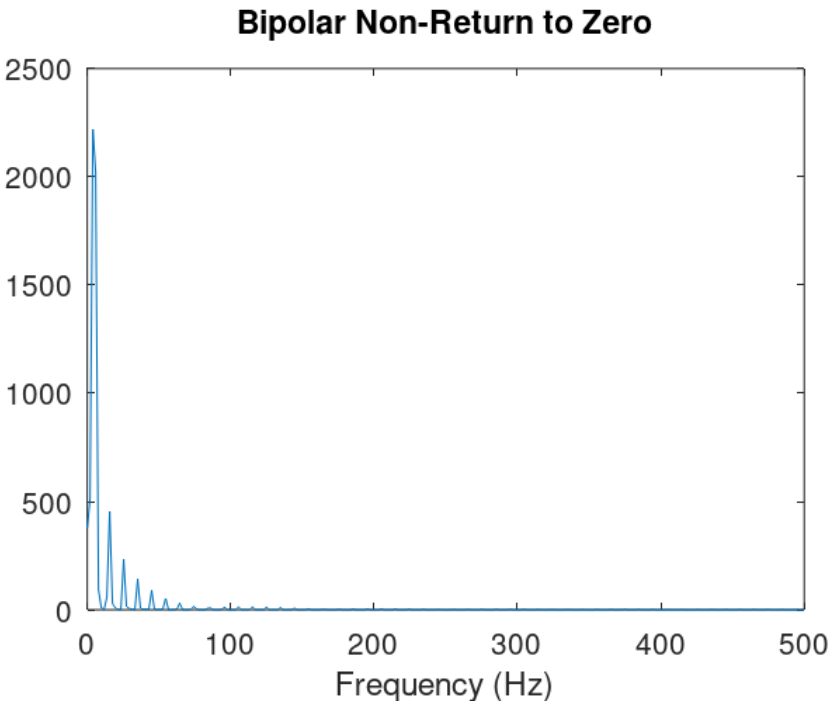


Рисунок 2.47: Кодирование NRZ: спектр сигнала

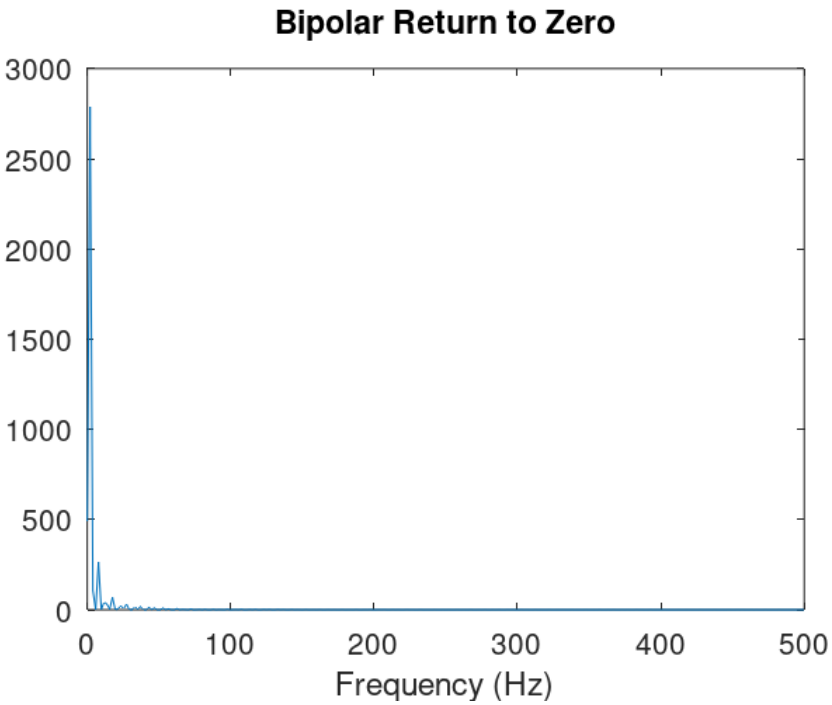


Рисунок 2.48: Кодирование RZ: спектр сигнала

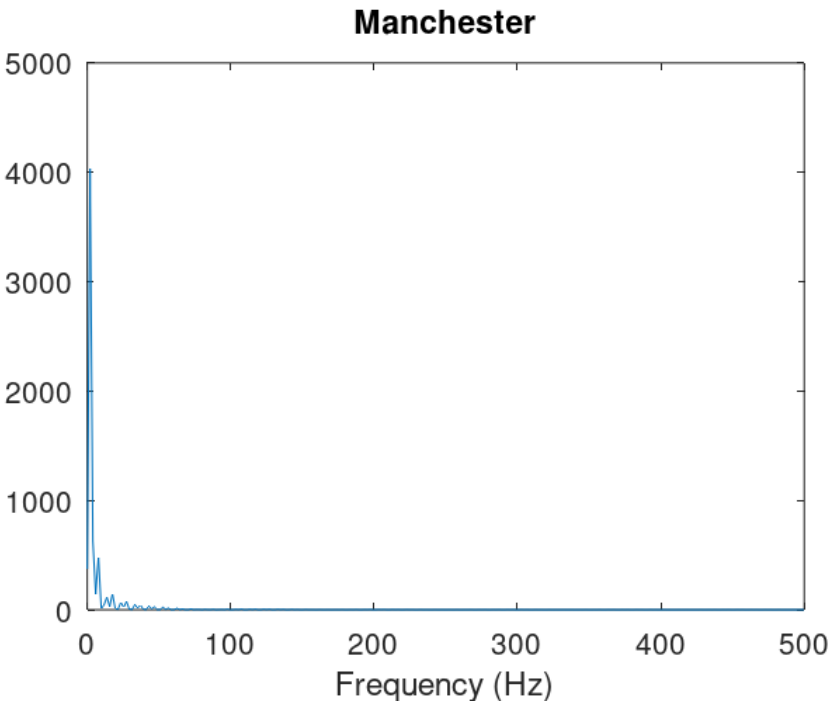


Рисунок 2.49: Манчестерское кодирование: спектр сигнала

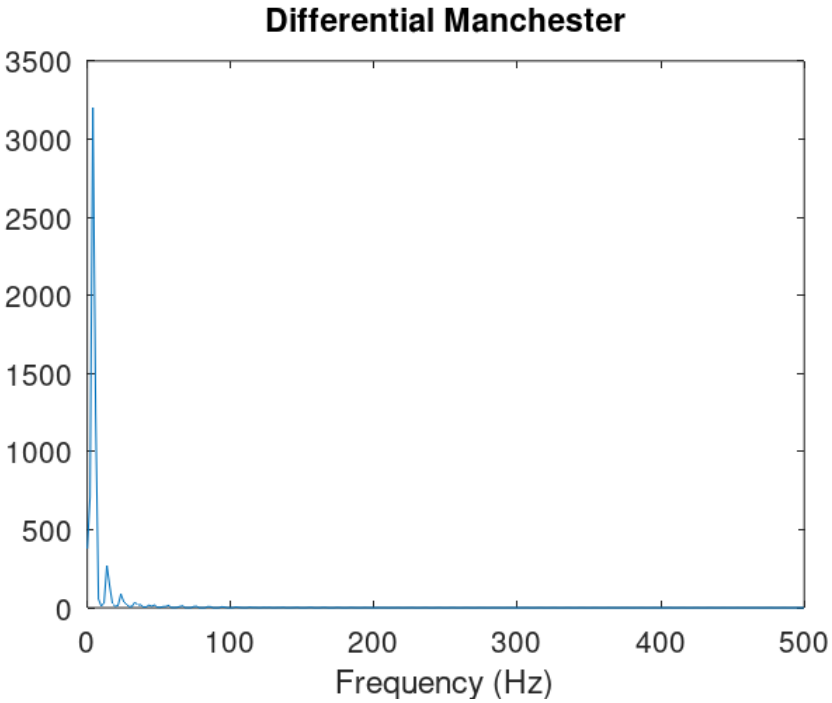


Рисунок 2.50: Дифференциальное манчестерское кодирование: спектр сигнала

3 Выводы

В ходе выполнения данной лабораторной работы я изучила методы кодирования и модуляции сигналов с помощью высокоуровнего языка программирования Octave. Определила спектр и параметры сигнала. Продемонстрировала принципы модуляции сигнала на примере аналоговой амплитудной модуляции. Исследовала свойства самосинхронизации сигнала.