Отчёт по лабораторной работе №1

Сетевые технологии

Мурашов Иван Вячеславович

Содержание

Список иллюстраций

Список Таблиц

1 Цель работы

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 Задание

1. Построить графики в Octave
    1. Построить график функции y = sin x + 1/3 * sin 3x + 1/5 * sin 5x на интервале [-10; 10], используя Octave и в файлы формата eps, .png.
    2. Добавить график функции y = cos x + 1/3 * cos 3x + 1/5 * cos 5x на интервале [-10; 10]. График экспортировать .
2. Разложение импульсного сигнала в частичный ряд Фурье: Разработать код m-файла, результатом выполнения которого являются графики меандра (рис. 1.3), реализованные с различным количеством гармоник.
3. Определение спектра и параметров сигнала
    1. Определить спектр двух отдельных сигналов и их суммы.
    2. Выполнить задание с другой частотой дискретизации. Пояснить, что будет, если взять частоту дискретизации меньше 80 Гц?
4. Амплитудная модуляция: Продемонстрировать принципы модуляции сигнала на примере аналоговой амплитудной модуляции (рис.1.9).

3 Выполнение лабораторной работы

3.1 Построение графиков в Octave

Запускаю Octave с оконным интерфейсом. Перехожу в окно редактора. Создаю новый сценарий. В окне редактора повторяю следующий листинг по построению графика функции:

```
% ▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ x:
x=-10:0.1:10;
% ▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ y.
y1=sin(x)+1/3*sin(3*x)+1/5*sin(5*x);
% ▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯:
plot(x,y1, "-ok; y1=sin(x)+
▯ (1/3)*sin(3*x)+(1/5)*sin(5*x);","markersize",4)
% ▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯ ▯▯ ▯▯▯▯▯▯▯▯
grid on;
% ▯▯▯▯▯▯▯▯ ▯▯▯▯ X:
xlabel('x');
% ▯▯▯▯▯▯▯▯ ▯▯▯▯ Y:
ylabel('y');
% ▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯:
title('y1=sin x+ (1/3)sin(3x)+(1/5)sin(5x)');
% ▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯ ▯ ▯▯▯▯ .eps:
```

```
print ("plot-sin.eps", "-mono", "-FArial:16", "-deps")
% □□□□□□□ □□□□□□□ □ □□□□ .png:
print("plot-sin.png");
```

После запуска мы видим созданный график, а в каталоге можем заметить файлы .png и .eps (рис. [**fig:001**]).

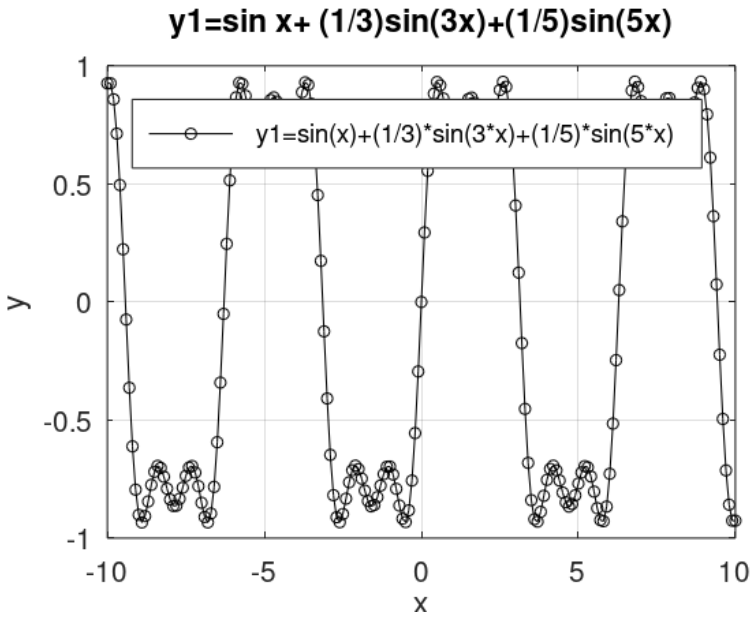

Рисунок 3.1: График функции

Сохраняю сценарий под другим названием и измените его так, чтобы на одном графике располагались отличающиеся по типу линий графики функций y1 = sin x + 1/3 * sin 3x + 1/5 * sin 5x и y2 = cos x + 1/3 * cos 3x + 1/5 * cos 5x. Соответственным образом модифицирую код:

```
% □□□□□□□□□□□□ □□□□□□□ x:
x=-10:0.1:10;
% □□□□□□□□□□□□ □□□□□□□ y.
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);
```

```
% [illegible]:
plot(x,y1, "-ok; y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);","markersi
hold on
plot(x,y2, "-; y2=cos(x)+1/3*cos(3*x)+1/5*cos(5*x);","markersize",4)
% [illegible]
grid on;
% [illegible] X:
xlabel('x');
% [illegible] Y:
ylabel('y');
% [illegible]:
% title('y1=sin x+ (1/3)sin(3x)+(1/5)sin(5x)');
% [illegible] .eps:
print ("plot-sin-cos.eps", "-mono", "-FArial:16", "-
deps")
% [illegible] .png:
print("plot-sin-cos.png");
```

Имеем график (рис. [**fig:002**]).

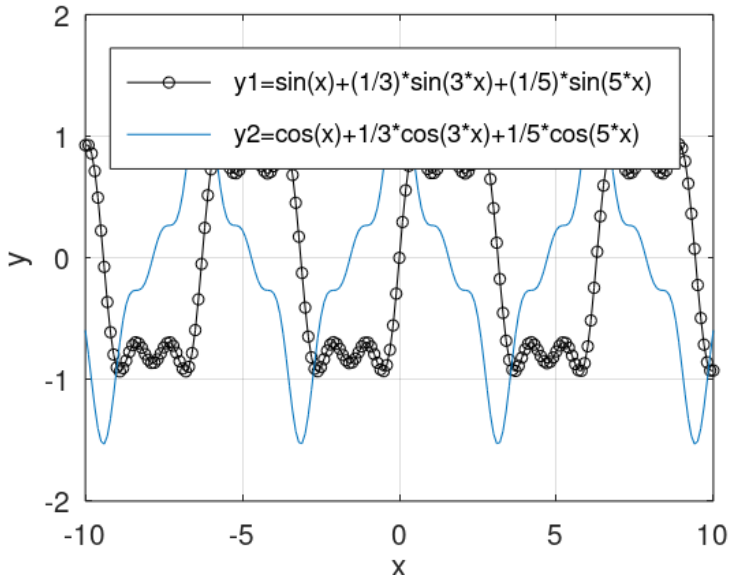

Рисунок 3.2: График функций 1 и 2 на интервале от−10 до 10 включая концы‘

3.2 Разложение импульсного сигнала в частичный ряд Фурье

Создаю файл meandr.m. В коде созданного сценария задаю начальные значения:

```
% meandr.m
% [illegible] [illegible] ([illegible]):
N=8;
% [illegible] [illegible]:
t=-1:0.01:1;
% [illegible] [illegible]:
A=1;
% [illegible]:
T=1;
```

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

```
% [illegible]
nh=(1:N)*2-1;
% [illegible], [illegible] cos:
Am=2/pi ./ nh;
Am(2:2:end) = -Am(2:2:end);
```

Далее задаём массив значений гармоник массив элементов ряда:

```
% [illegible]:
harmonics=cos(2 * pi * nh' * t/T);
% [illegible]:
s1=harmonics.*repmat(Am',1,length(t));
```

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:

```
% [illegible]:
s2=cumsum(s1);
% [illegible]:
for k=1:N
subplot(4,2,k)
plot(t, s2(k,:))
end
```

Экспортируем полученный график в файл в формате .png (рис. [**fig:003**]).

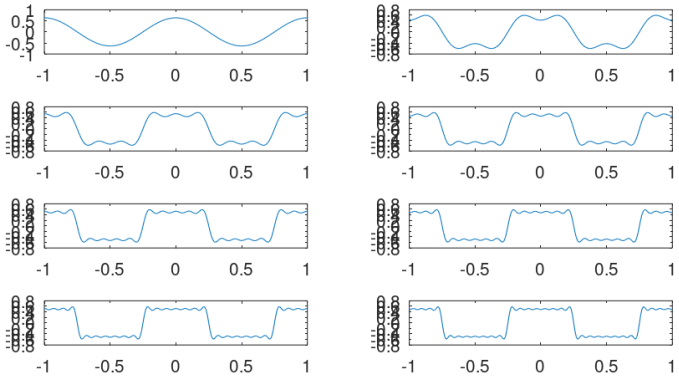

Рисунок 3.3: Графики меандра, содержащего различное число гармоник

Корректируем код для реализации меандра через синусы:

```
% meandr.m
% [illegible] [illegible] ([illegible]):
N=8;
% [illegible] [illegible]:
t=-1:0.01:1;
% [illegible] [illegible]:
A=1;
% [illegible]:
T=1;
% [illegible] [illegible]
nh=(1:N)*2-1;
% [illegible] [illegible] [illegible], [illegible] sin:
Am=2/pi ./ nh;
% Am(2:2:end) = -Am(2:2:end); -not needed
% [illegible] [illegible]:
harmonics=sin(2 * pi * nh' * t/T);
% [illegible] [illegible]:
s1=harmonics.*repmat(Am',1,length(t));
```

```
% □□□□□□□□□□□□□ □□□□:
s2=cumsum(s1);
% □□□□□□□□□□ □□□□□□□□:
for k=1:N
subplot(4,2,k)
plot(t, s2(k,:))
end

% □□□□□□□ □ .png
print("plot-meandr-sin.png")
```

Получаем аналогичный график (рис. [**fig:004**]).

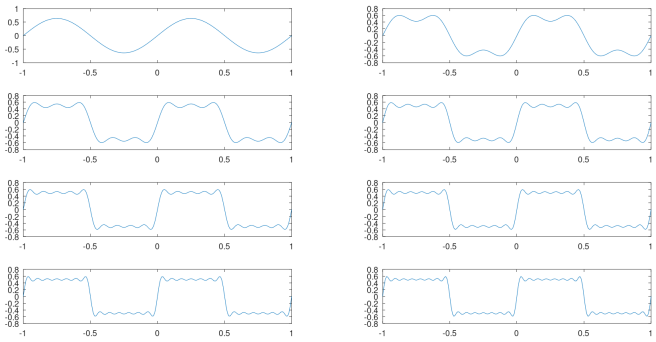

Рисунок 3.4: Графики меандра, содержащего различное число гармоник

3.3 Определение спектра и параметров сигнала

В рабочем каталоге создаю каталог spectre1 и в нём новый сценарий с именем, spectre.m. В коде созданного сценария задайте начальные значения:

```
% spectre1/spectre.m
% □□□□□□□□ □□□□□□□□□□ signal □ spectre □□□ □□□□□□□□□□
□ □□□□□□□□:
```

```
mkdir 'signal';
mkdir 'spectre';
% [illegible] [illegible] (с):
tmax = 0.5;
% [illegible] [illegible] (Гц) ([illegible] [illegible]):
fd = 512;
% [illegible] [illegible] [illegible] (Гц):
f1 = 10;
% [illegible] [illegible] [illegible] (Гц):
f2 = 40;
% [illegible] [illegible] [illegible]:
a1 = 1;
% [illegible] [illegible] [illegible]:
a2 = 0.7;
% [illegible] [illegible] [illegible]:
t = 0:1./fd:tmax;
% [illegible] [illegible]:
fd2 = fd/2;
```

Далее в коде задаю два синусоидальных сигнала разной частоты:

```
% [illegible] [illegible] [illegible] [illegible]:
signal1 = a1*sin(2*pi*t*f1);
signal2 = a2*sin(2*pi*t*f2);
```

Строю графики сигналов (рис. [**fig:005**]).

```
% [illegible] 1-[illegible] [illegible]:
plot(signal1,'b');
% [illegible] 2-[illegible] [illegible]:
hold on
```

```
plot(signal2,'r');
hold off
title('Signal');
% [illegible] [illegible] [illegible] [illegible] signal:
print 'signal/spectre.png';
```

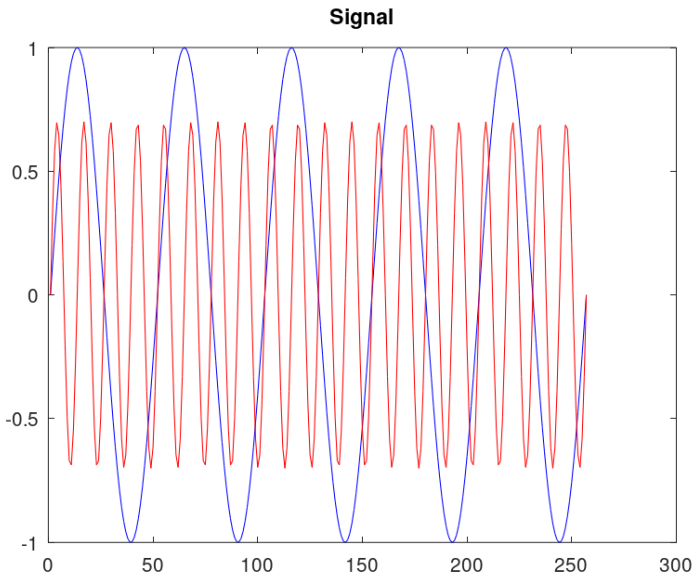

Рисунок 3.5: Два синусоидальных сигнала разной частоты

С помощью быстрого преобразования Фурье находим спектры сигналов (рис. [**fig:006**]), добавив в файл spectre.m следующий код:

```
% [illegible] [illegible]
% [illegible] [illegible] [illegible] [illegible] 1:
spectre1 = abs(fft(signal1,fd));
% [illegible] [illegible] [illegible] [illegible] 2:
spectre2 = abs(fft(signal2,fd));
% [illegible] [illegible] [illegible] [illegible]:
plot(spectre1,'b');
hold on
```

```
plot(spectre2,'r');
hold off
title('Spectre');
print 'spectre/spectre.png';
```

Учитывая реализацию преобразования Фурье, корректирую график спектра (рис. [**fig:007**]): отбросываем дублирующие отрицательные частоты, а также принимаем в расчёт то, что на каждом шаге вычисления быстрого преобразования Фурье происходит суммирование амплитуд сигналов. Для этого добавляю в файл spectre.m следующий код:

```
% [illegible]
% [illegible]:
f = 1000*(0:fd2)./(2*fd);
% [illegible] [illegible]:
spectre1 = 2*spectre1/fd2;
spectre2 = 2*spectre2/fd2;
% [illegible]:
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';
```

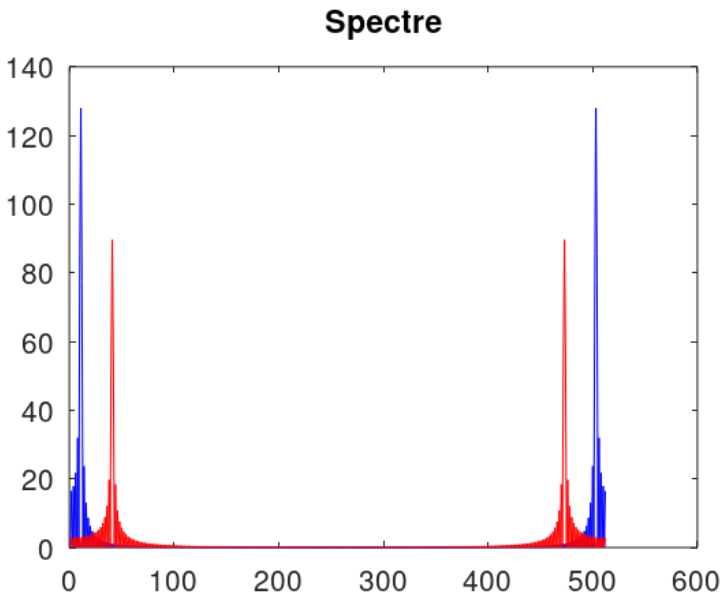

Рисунок 3.6: График спектров синусоидальных сигналов

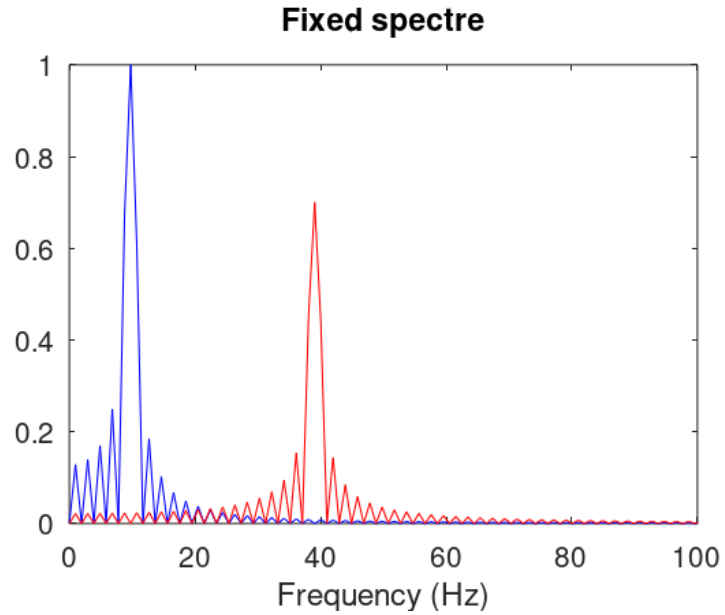

Рисунок 3.7: Исправленный график спектров синусоидальных сигналов

Найдём спектр суммы рассмотренных сигналов (рис. [**fig:008**]), создав каталог spectr_sum и файл в нём spectre_sum.m со следующим кодом:

ctr_sum spectre_sum.m :

```
% spectr_sum/spectre_sum.m
% [illegible] signal [illegible] spectre [illegible]
[illegible]:
mkdir 'signal';
mkdir 'spectre';
% [illegible] [illegible] (c):
tmax = 0.5;
% [illegible] [illegible] (Hz) ([illegible] [illegible]):
fd = 512;
% [illegible] [illegible] (Hz):
f1 = 10;
% [illegible] [illegible] (Hz):
f2 = 40;
% [illegible] [illegible]:
a1 = 1;
% [illegible] [illegible]:
a2 = 0.7;
% [illegible]
fd2 = fd/2;
% [illegible] [illegible] ([illegible]) [illegible]:
% [illegible] [illegible]:
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';
% [illegible]:
```

```
% [illegible]:
spectre = fft(signal,fd);
% [illegible]
f = 1000*(0:fd2)./(2*fd);
% [illegible]:
spectre = 2*sqrt(spectre.*conj(spectre))./fd2;
% [illegible]:
plot(f,spectre(1:fd2+1))
xlim([0 100]);
title('Spectre');
xlabel('Frequency (Hz)');
print 'spectre/spectre_sum.png';
```

В результате получим аналогичный предыдущему результат (рис. [**fig:009**]), т.е. спектр суммы сигналов равен сумме спектров сигналов, что вытекает из свойств преобразования Фурье.

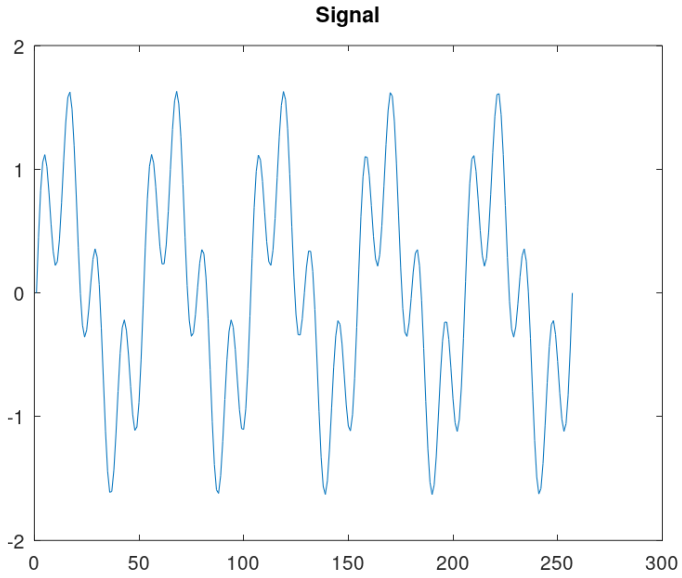

Рисунок 3.8: Суммарный сигнал

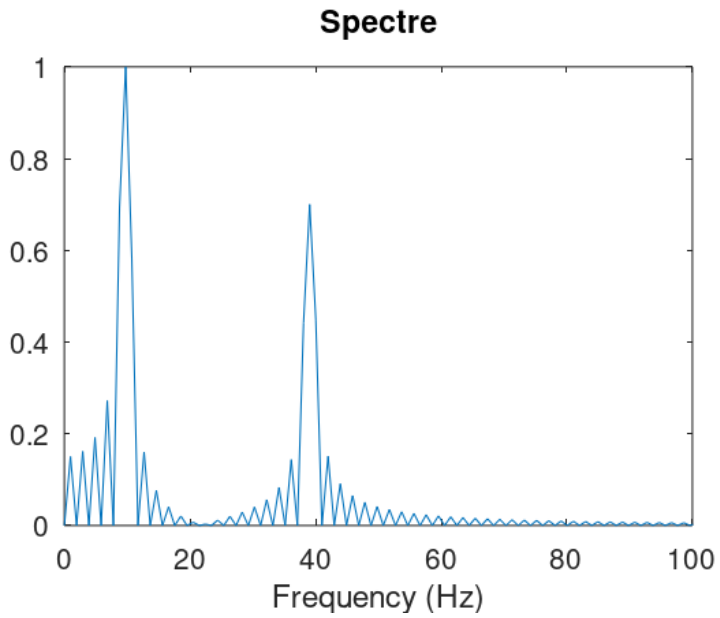

Рисунок 3.9: Спектр суммарного сигнала

3.4 Амплитудная модуляция

Создаю каталог modulation и в нём новый сценарий с именем am.m. Добавляю в файле am.m следующий код:

```
% modulation/am.m
% □□□□□□□□ □□□□□□□□□ signal □ spectre □□□ □□□□□□□□□□
□ □□□□□□□□:
mkdir 'signal';
mkdir 'spectre';
% □□□□□□□□□ □□□□□□□□ □ □□□□□□□□□ 50 □ 5
% □□□□□ □□□□□□□ (□)
tmax = 0.5;
% □□□□□□□ □□□□□□□□□□□□□ (□□) (□□□□□□□□□□ □□□□□□□□)
fd = 512;
% □□□□□□□ □□□□□□□ (□□)
```

```
f1 = 5;
% [illegible] [illegible] ([illegible])
f2 = 50;
% [illegible] [illegible]
fd2 = fd/2;
% [illegible] [illegible] [illegible] [illegible] ([illegible])
% [illegible] [illegible]
% [illegible] [illegible] [illegible]:
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
% [illegible] [illegible]:
plot(signal1, 'r');
plot(-signal1, 'r');
hold off
title('Signal');
print 'signal/am.png';
% [illegible] [illegible]:
% [illegible] [illegible] [illegible]-[illegible]:
spectre = fft(signal,fd);
% [illegible] [illegible]:
f = 1000*(0:fd2)./(2*fd);
% [illegible] [illegible] [illegible] [illegible]:
spectre = 2*sqrt(spectre.*conj(spectre))./fd2;
% [illegible] [illegible]:
plot(f,spectre(1:fd2+1), 'b')
```

```
xlim([0 100]);
title('Spectre');
xlabel('Frequency (Hz)');
print 'spectre/am.png';
```

В результате получаем, что спектр произведения представляет собой свёртку спектров (рис. [**fig:011**]).

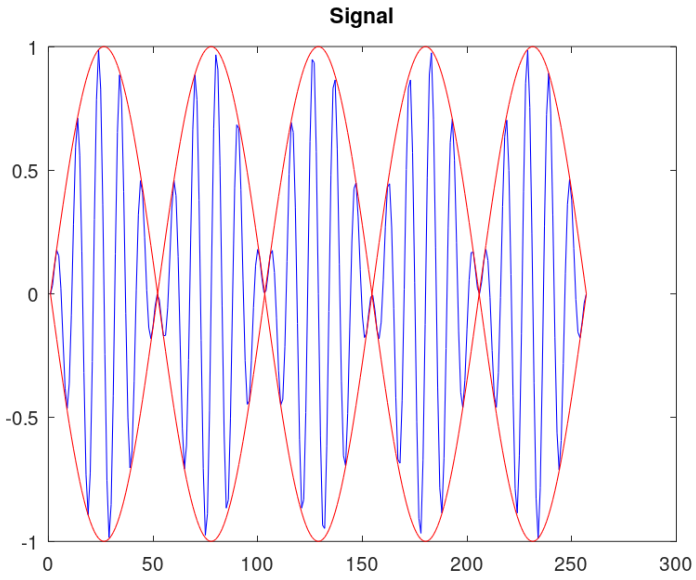

Рисунок 3.10: Сигнал и огибающая при амплитудной модуляции

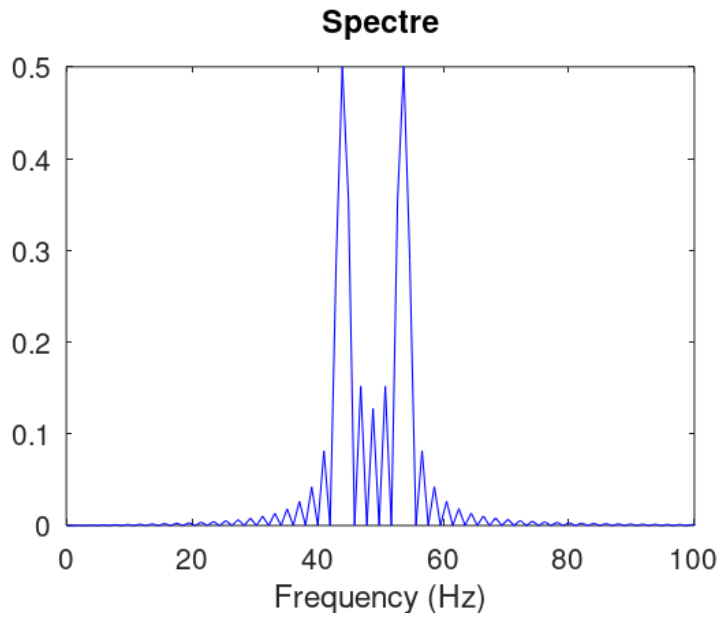

Рисунок 3.11: Спектр сигнала при амплитудной модуляции

3.5 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

Создаю 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 файле main.m подключаем пакет signal и задаём входные кодовые последовательности:

```
% coding/main.m
% [illegible] [illegible] signal:
pkg load signal;

% [illegible] [illegible] [illegible]:
data=[0 1 0 0 1 1 0 0 0 1 1 0];
% [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible] [illegible]:
```

```
data_sync=[0 0 0 0 0 0 0 1 1 1 1 1 1 1];
% ▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯▯
▯ ▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯:
data_spectre=[0 1 0 1 0 1 0 1 0 1 0 1 0 1];
% ▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯ signal, sync ▯ spectre ▯▯▯▯
▯ ▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯:
mkdir 'signal';
mkdir 'sync';
mkdir 'spectre';
axis("auto");
```

Затем в этом же файле прописываем вызовы функций для построения графиков модуляций кодированных сигналов для кодовой последовательности data:

```
% ▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯▯▯▯▯▯
wave=unipolar(data);
plot(wave);
ylim([-1 6]);
title('Unipolar');
print 'signal/unipolar.png';
% ▯▯▯▯▯▯▯▯▯▯▯▯ ami
wave=ami(data);
plot(wave)
title('AMI');
print 'signal/ami.png';
% ▯▯▯▯▯▯▯▯▯▯▯▯ NRZ
wave=bipolarnrz(data);
plot(wave);
title('Bipolar Non-Return to Zero');
print 'signal/bipolarnrz.png';
```

```
% [illegible] RZ
wave=bipolarrz(data);
plot(wave)
title('Bipolar Return to Zero');
print 'signal/bipolarrz.png';
% [illegible] [illegible]
wave=manchester(data);
plot(wave)
title('Manchester');
print 'signal/manchester.png';
% [illegible] [illegible] [illegible]
wave=diffmanc(data);
plot(wave)
title('Differential Manchester');
print 'signal/diffmanc.png';
```

Затем в этом же файле пропишите вызовы функций для построения графиков модуляций кодированных сигналов для кодовой последовательности data_sync:

```
% [illegible] [illegible]
wave=unipolar(data_sync);
plot(wave);
ylim([-1 6]);
title('Unipolar');
print 'sync/unipolar.png';
% [illegible] AMI
wave=ami(data_sync);
plot(wave)
title('AMI');
print 'sync/ami.png';
```

```
% □□□□□□□□□□□ NRZ
wave=bipolarnrz(data_sync);
plot(wave);
title('Bipolar Non-Return to Zero');
print 'sync/bipolarnrz.png';
% □□□□□□□□□□□ RZ
wave=bipolarrz(data_sync);
plot(wave)
title('Bipolar Return to Zero');
print 'sync/bipolarrz.png';
% □□□□□□□□□□□□□ □□□□□□□□□□□
wave=manchester(data_sync);
plot(wave)
title('Manchester');
print 'sync/manchester.png';
% □□□□□□□□□□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□
wave=diffmanc(data_sync);
plot(wave)
title('Differential Manchester');
print 'sync/diffmanc.png';
```

Далее в этом же файле прописываем вызовы функций для построения графиков спектров:

```
% □□□□□□□□□□□ □□□□□□□□□□□:
wave=unipolar(data_spectre);
spectre=calcspectre(wave);
title('Unipolar');
print 'spectre/unipolar.png';
% □□□□□□□□□□□ AMI:
```

```
wave=ami(data_spectre);
spectre=calcspectre(wave);
title('AMI');
print 'spectre/ami.png';
% [illegible] NRZ:
wave=bipolarnrz(data_spectre);
spectre=calcspectre(wave);
title('Bipolar Non-Return to Zero');
print 'spectre/bipolarnrz.png';
% [illegible] RZ:
wave=bipolarrz(data_spectre);
spectre=calcspectre(wave);
title('Bipolar Return to Zero');
print 'spectre/bipolarrz.png';
% [illegible] [illegible]:
wave=manchester(data_spectre);
spectre=calcspectre(wave);
title('Manchester');
print 'spectre/manchester.png';
% [illegible] [illegible] [illegible]:
wave=diffmanc(data_spectre);
spectre=calcspectre(wave);
title('Differential Manchester');
print 'spectre/diffmanc.png';
```

В файле maptowave.m прописываем 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);
```

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

Униполярное кодирование:

```
% coding/unipolar.m
% [illegible]:
function wave=unipolar(data)
wave=maptowave(data);
```

Кодирование AMI:

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

Кодирование NRZ:

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

Кодирование RZ:

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

Манчестерское кодирование:

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

Дифференциальное манчестерское кодирование:

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

В файле calcspectre.m прописываем функцию построения спектра сигнала:

```
% calcspectre.m
% [illegible]:
```

```
function spectre = calcspectre(wave)
% [illegible] [illegible] ([illegible]):
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:Fd3));
xlabel('Frequency (Hz)');
```

Запускаю главный скрипт main.m. В каталоге signal должны быть получены файлы с графиками кодированного сигнала (рис. [**fig:012**] – рис. [**fig:017**]), в каталоге sync — файлы с графиками, иллюстрирующими свойства самосинхронизации (рис. [**fig:018**] – рис. [**fig:023**]), в каталоге spectre — файлы с графиками спектров сигналов (рис. [**fig:024**] – рис. [**fig:029**]).

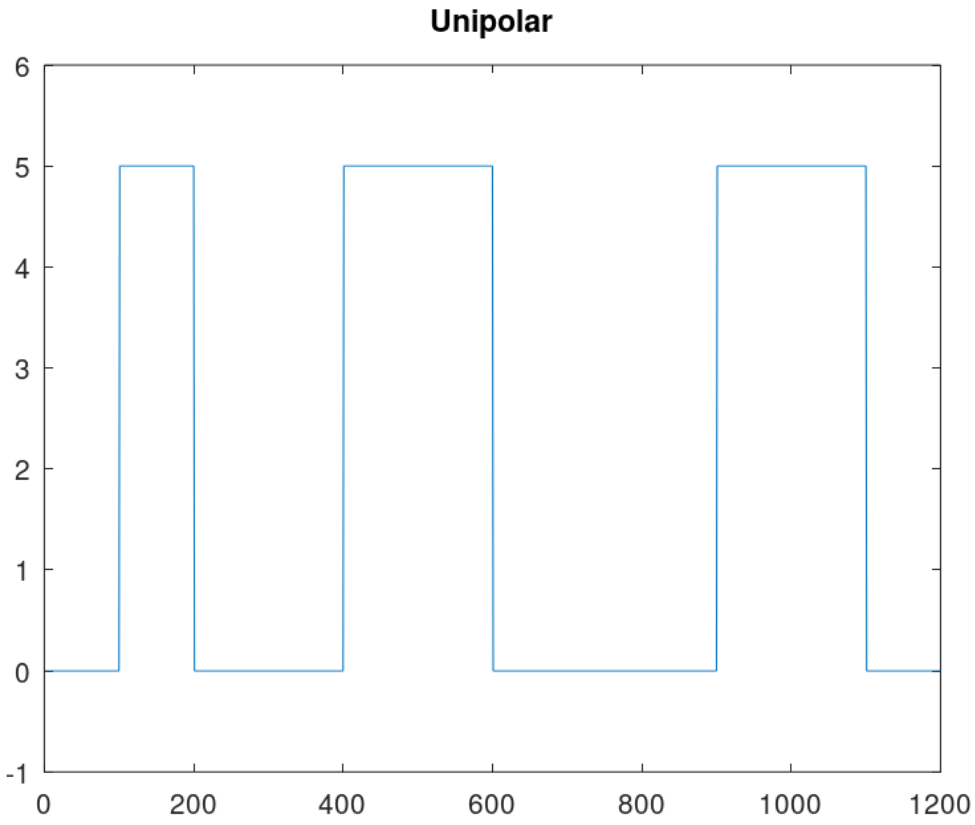

Рисунок 3.12: Униполярное кодирование

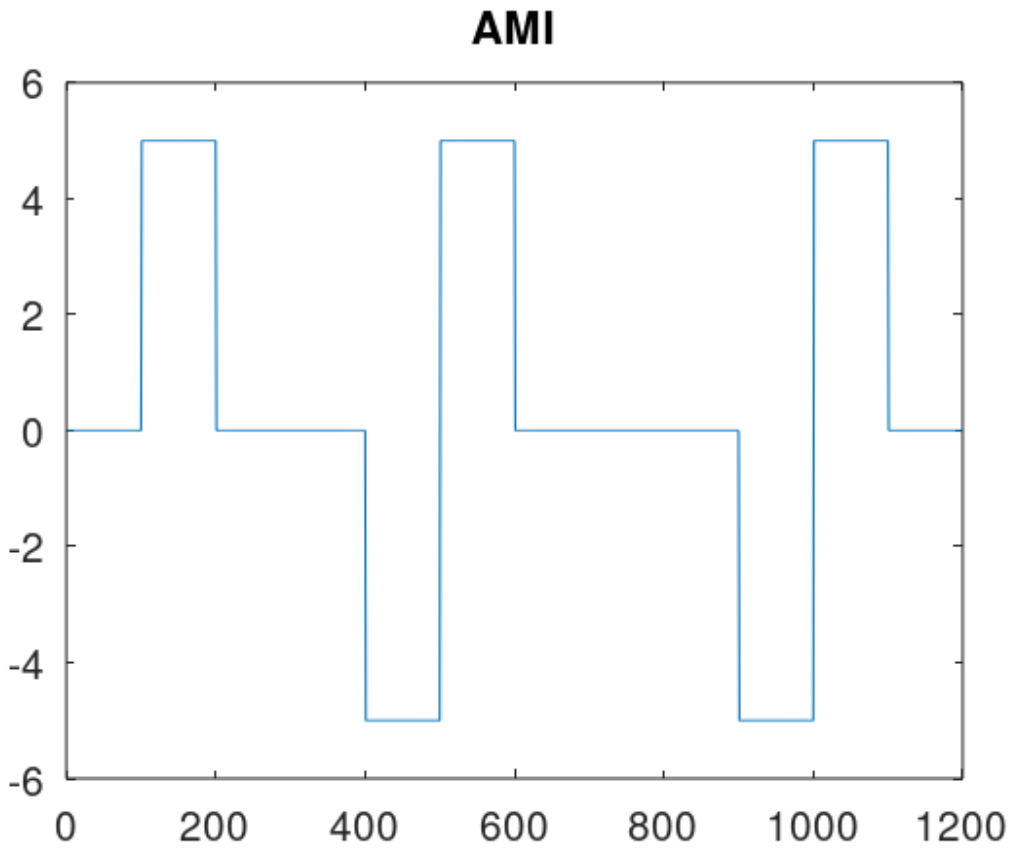

Рисунок 3.13: Кодирование AMI

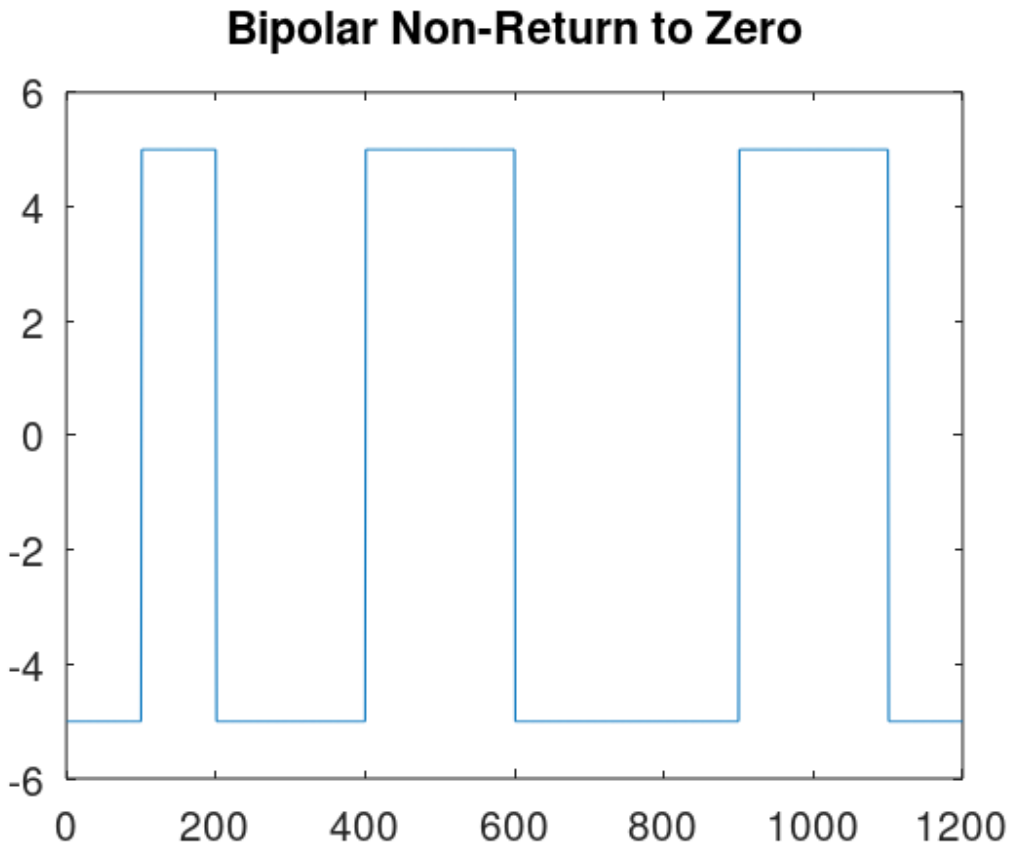

Рисунок 3.14: Кодирование NRZ

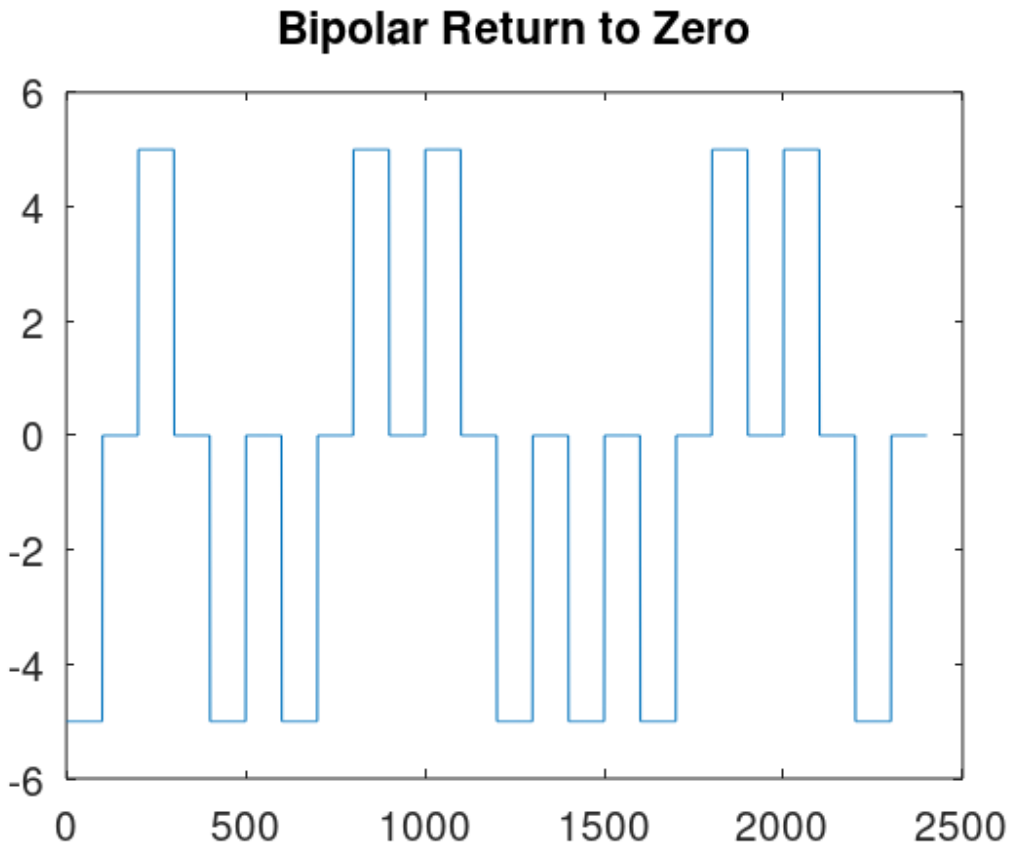

Рисунок 3.15: Кодирование RZ

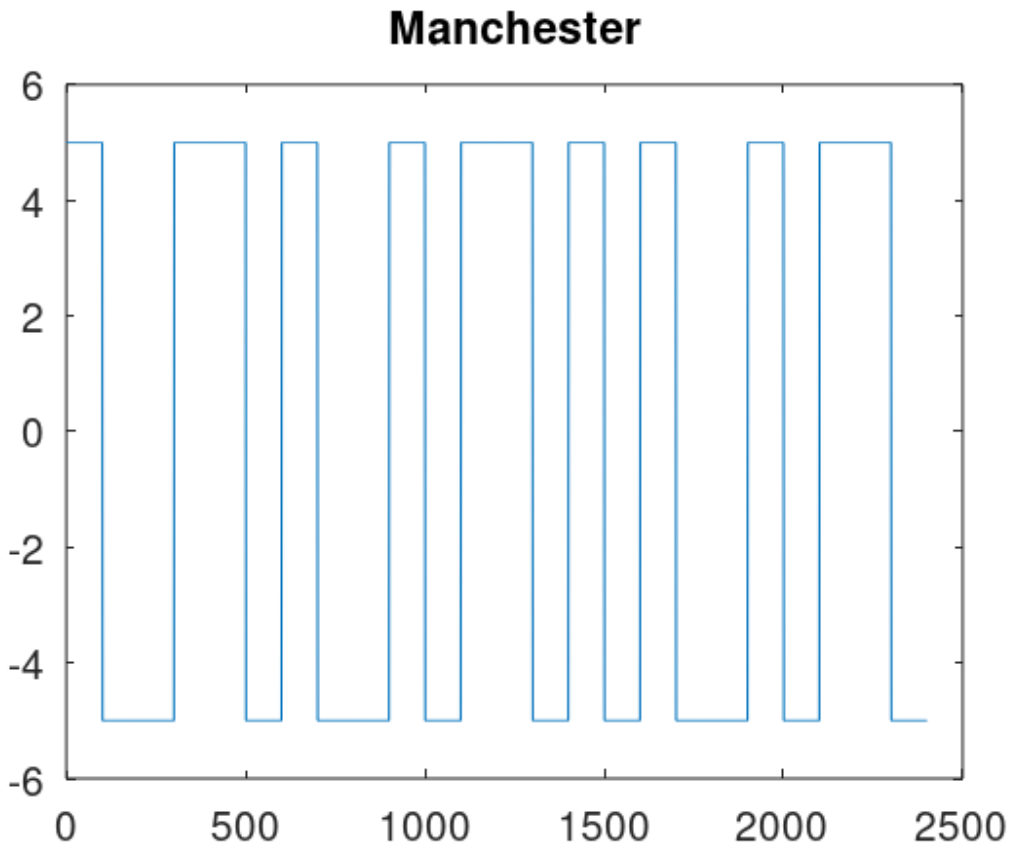

Рисунок 3.16: Манчестерское кодирование

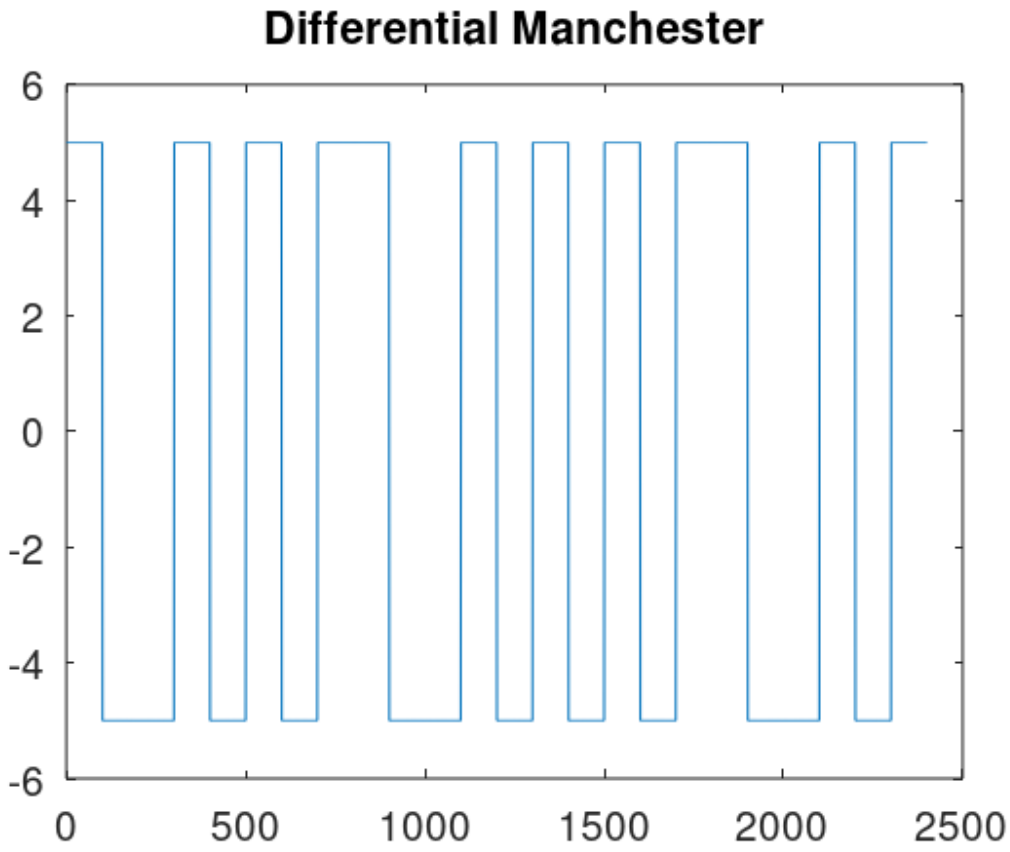

Рисунок 3.17: Дифференциальное манчестерское кодирование

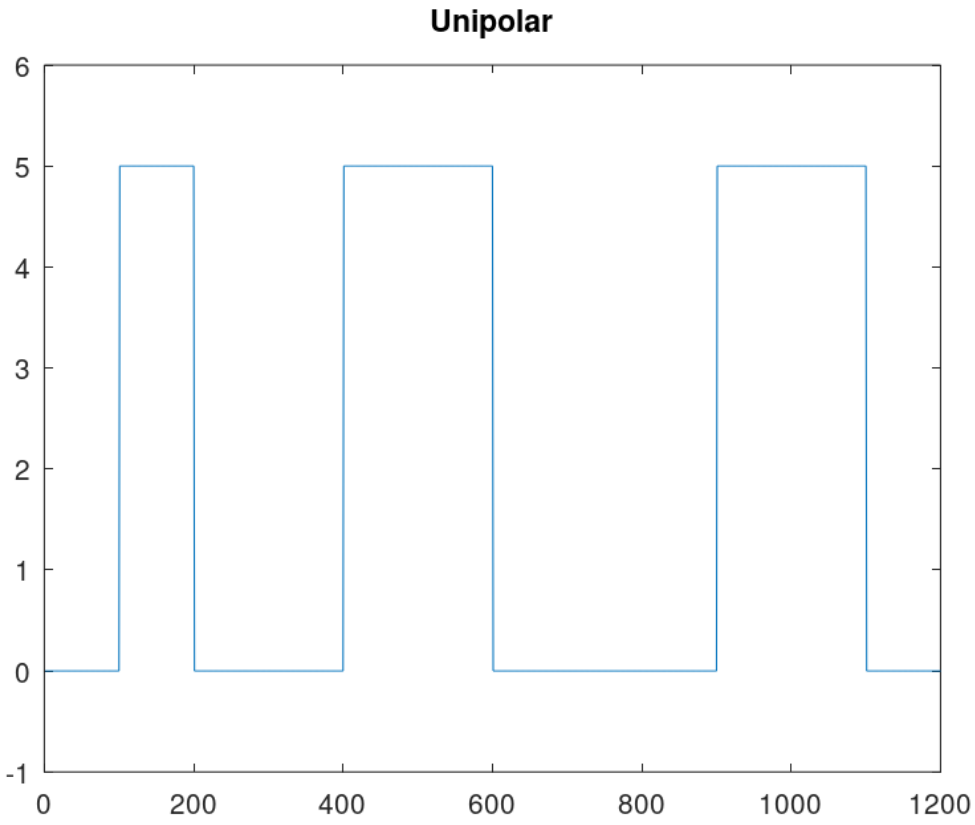

Рисунок 3.18: Униполярное кодирование: нет самосинхронизации

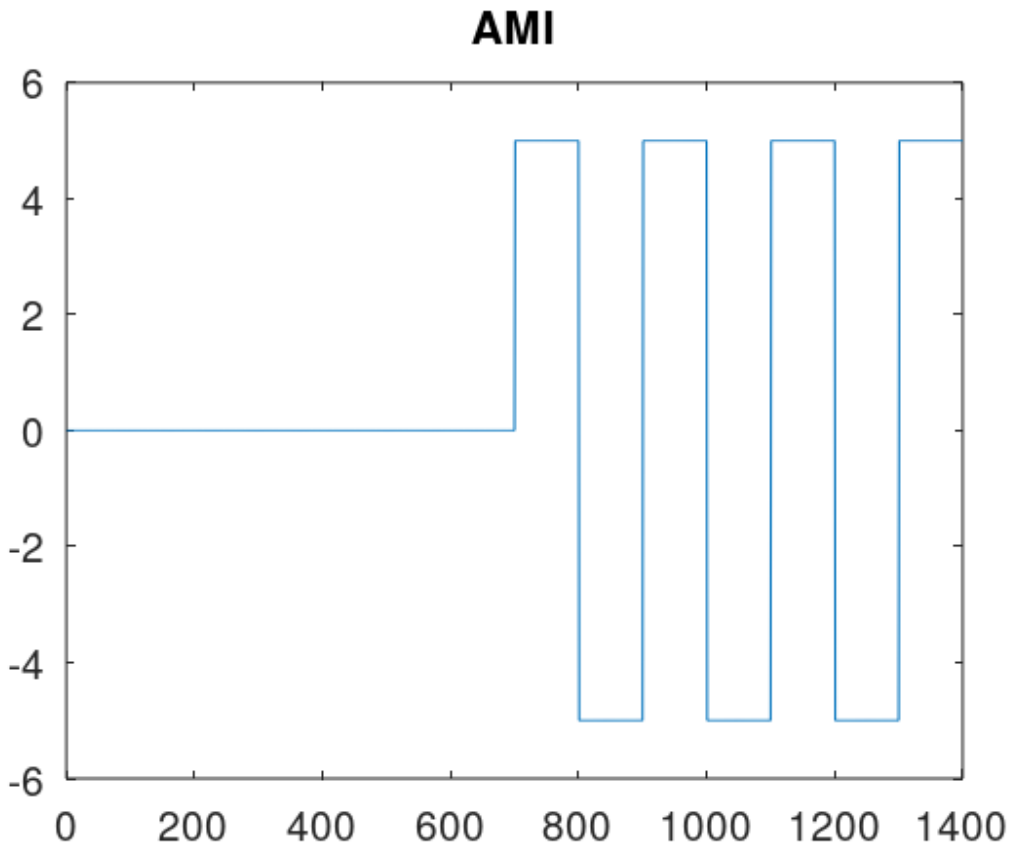

Рисунок 3.19: Кодирование AMI: самосинхронизация при наличии сигнала

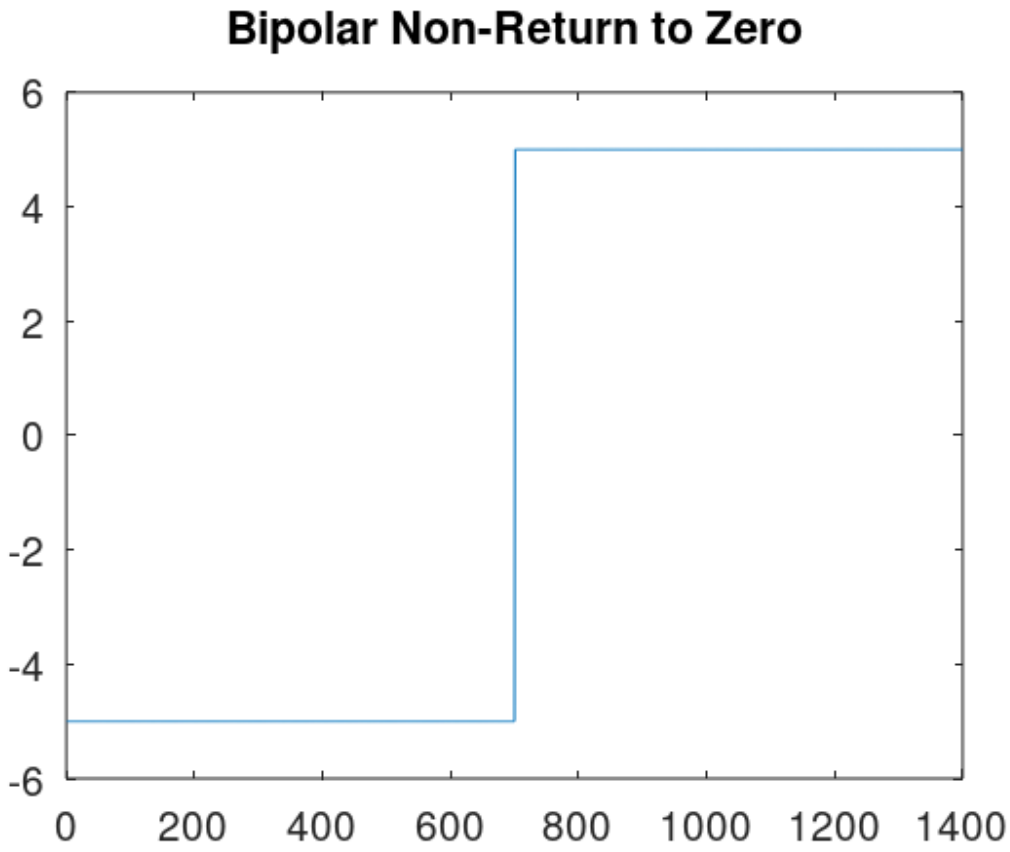

Рисунок 3.20: Кодирование NRZ: нет самосинхронизации

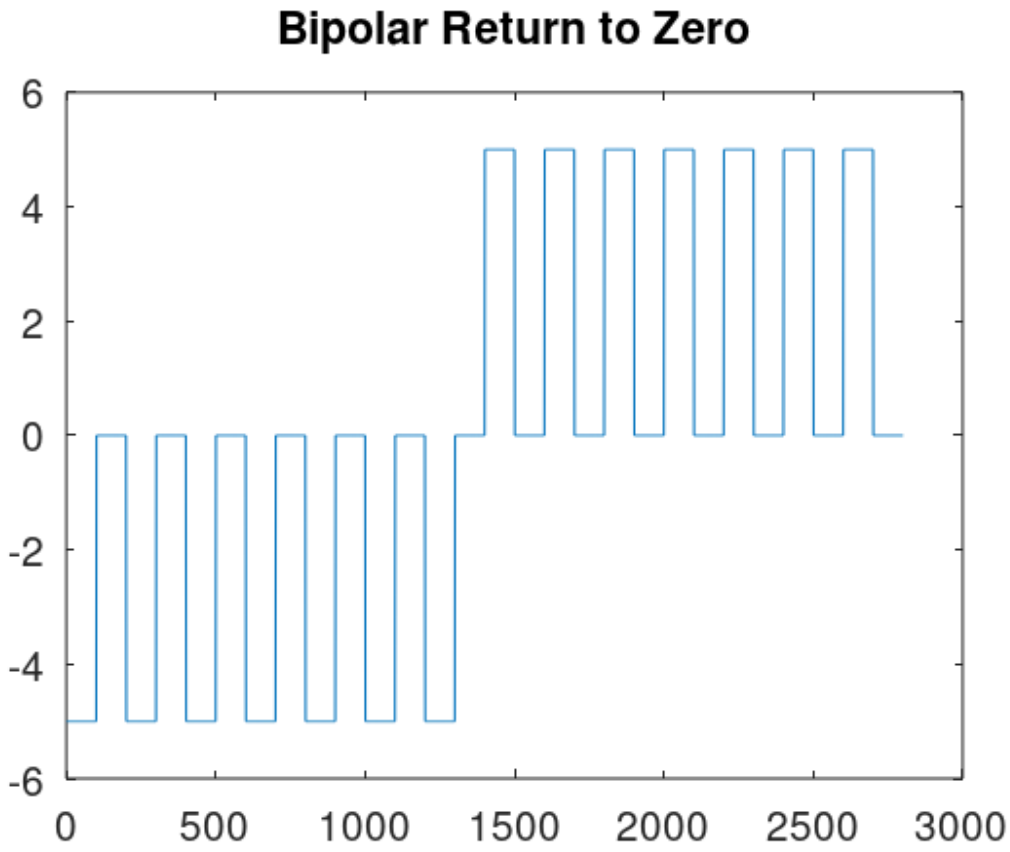

Рисунок 3.21: Кодирование RZ: есть самосинхронизация

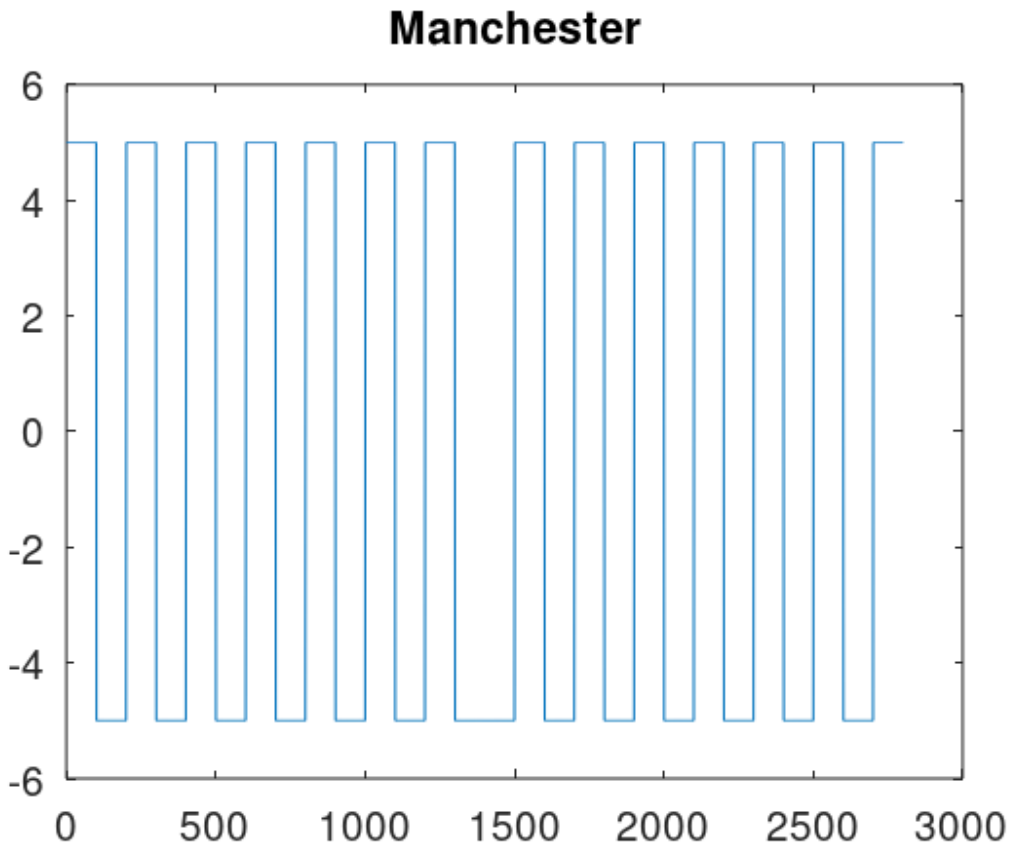

Рисунок 3.22: Манчестерское кодирование: есть самосинхронизация

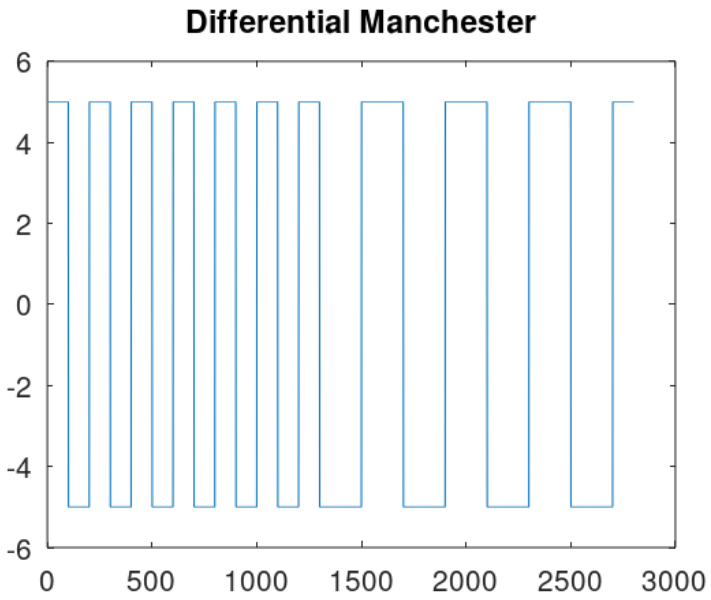

Рисунок 3.23: Дифференциальное манчестерское кодирование: есть самосинхронизация

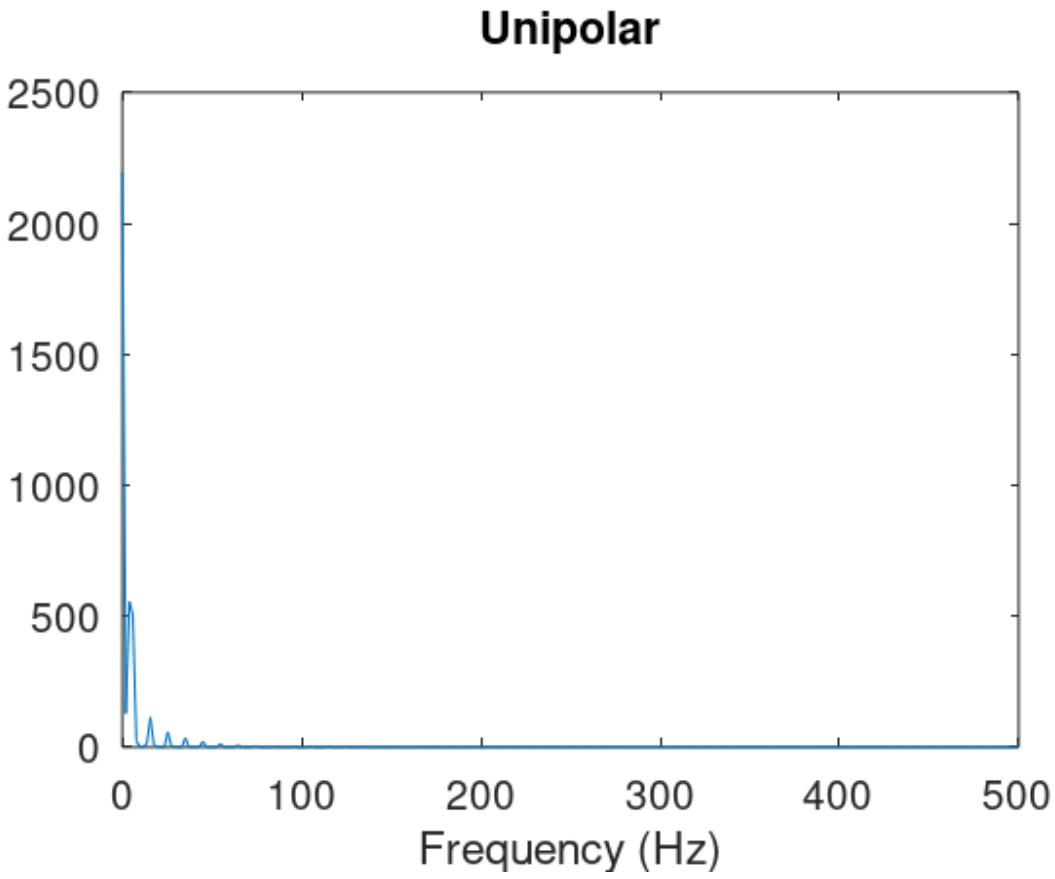

Рисунок 3.24: Униполярное кодирование: спектр сигнала

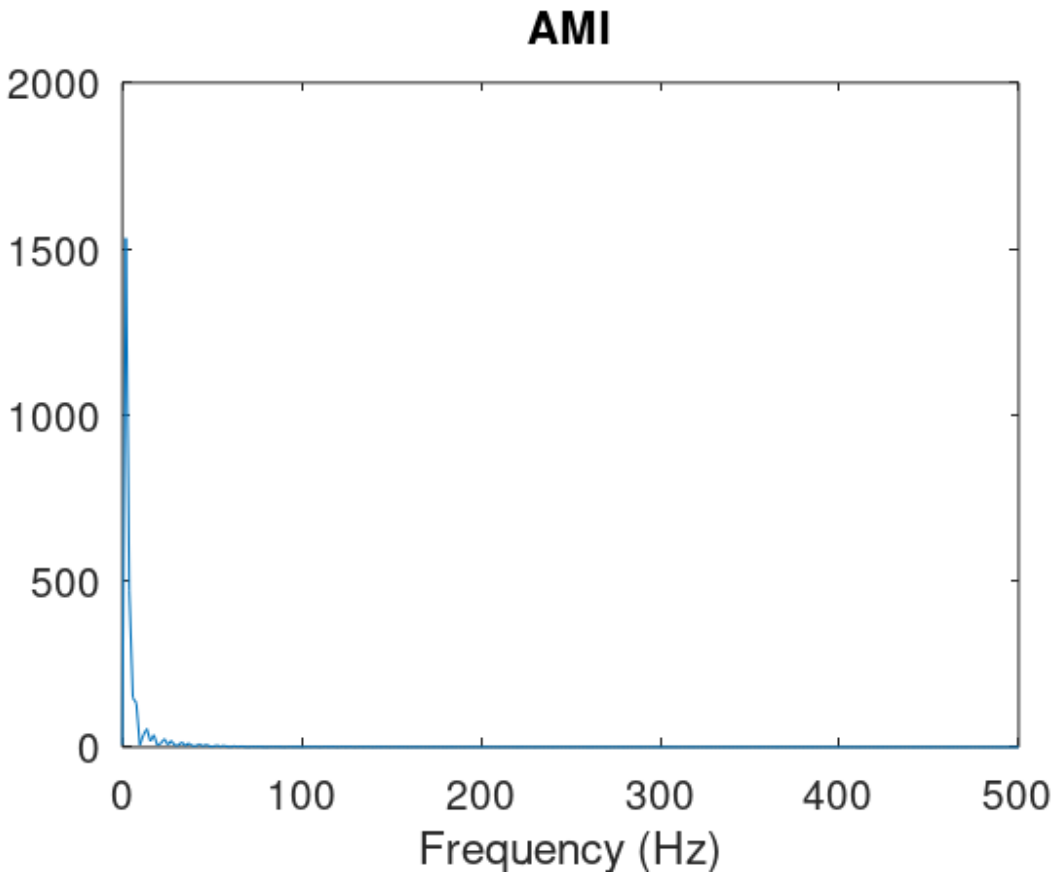

Рисунок 3.25: Кодирование AMI: спектр сигнала

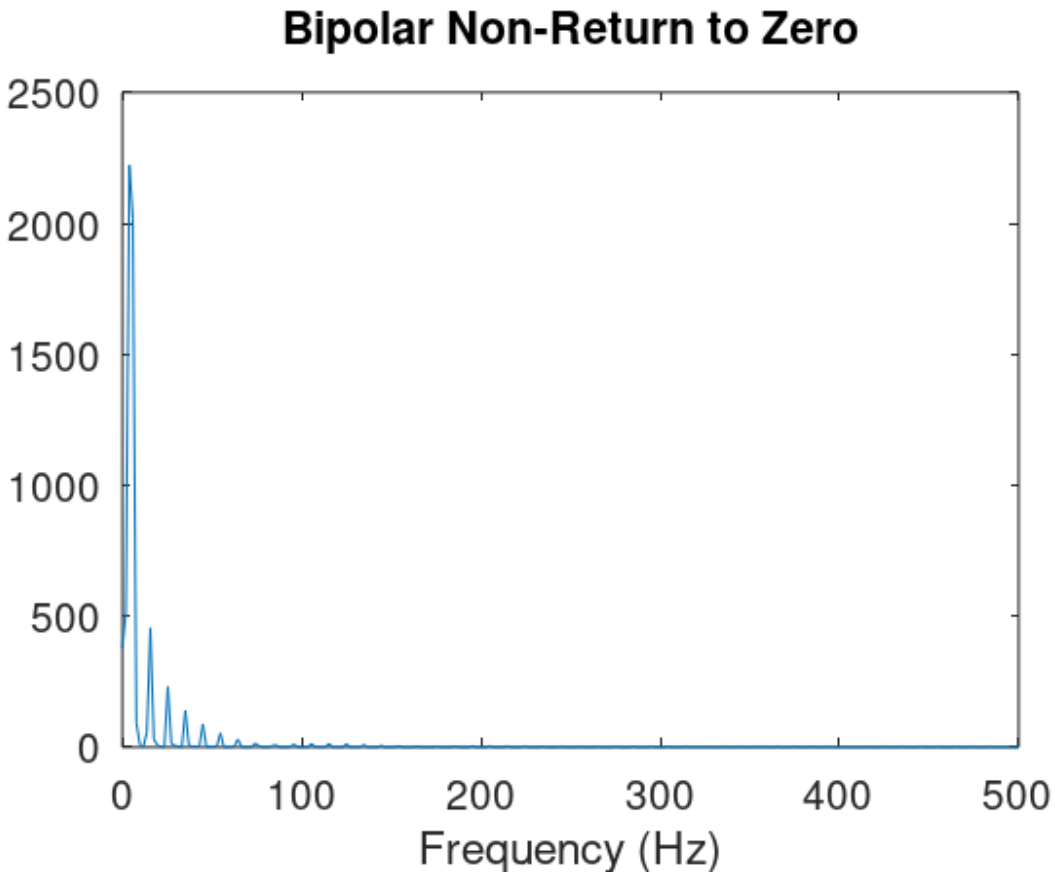

Рисунок 3.26: Кодирование NRZ: спектр сигнала

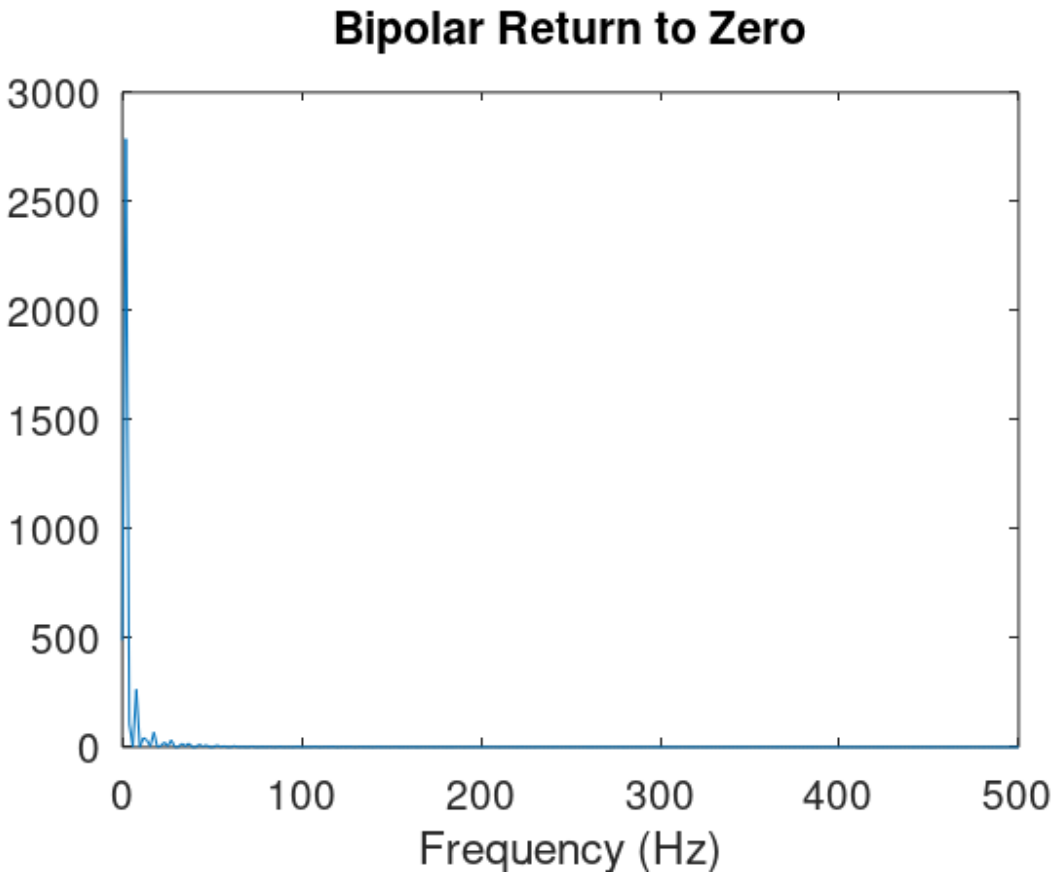

Рисунок 3.27: Кодирование RZ: спектр сигнала

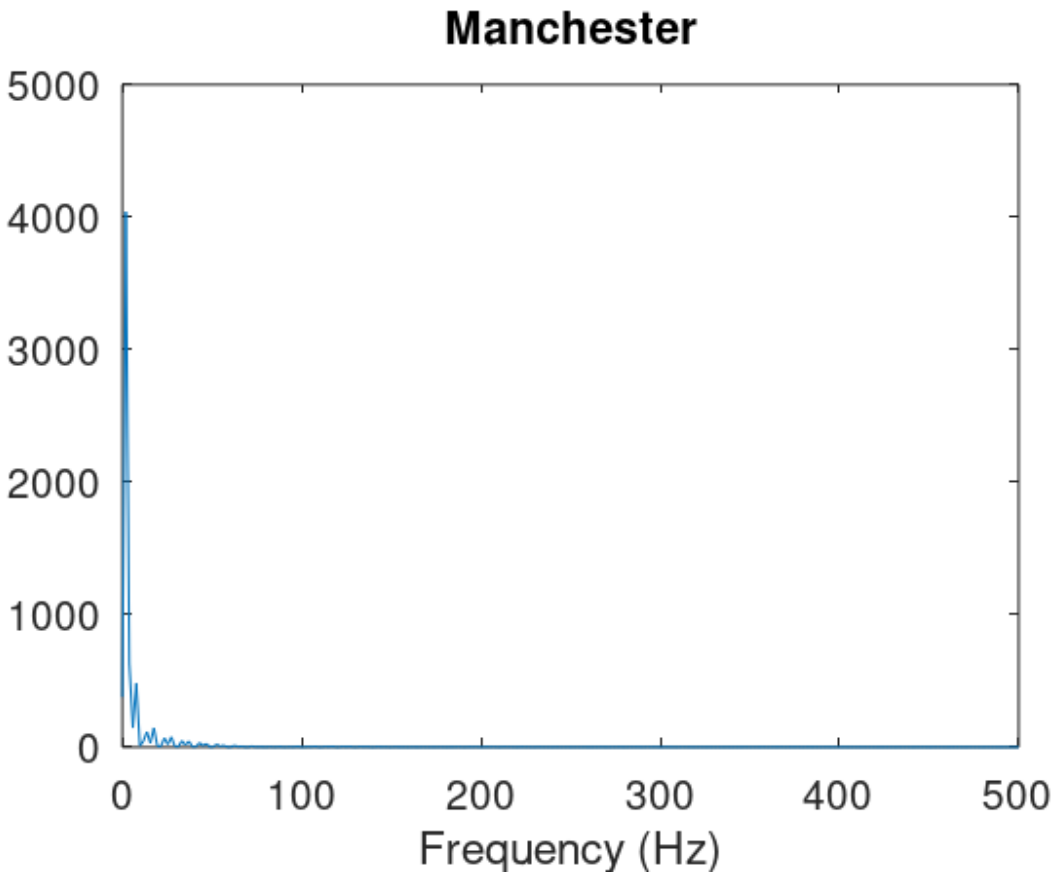

Рисунок 3.28: Манчестерское кодирование: спектр сигнала

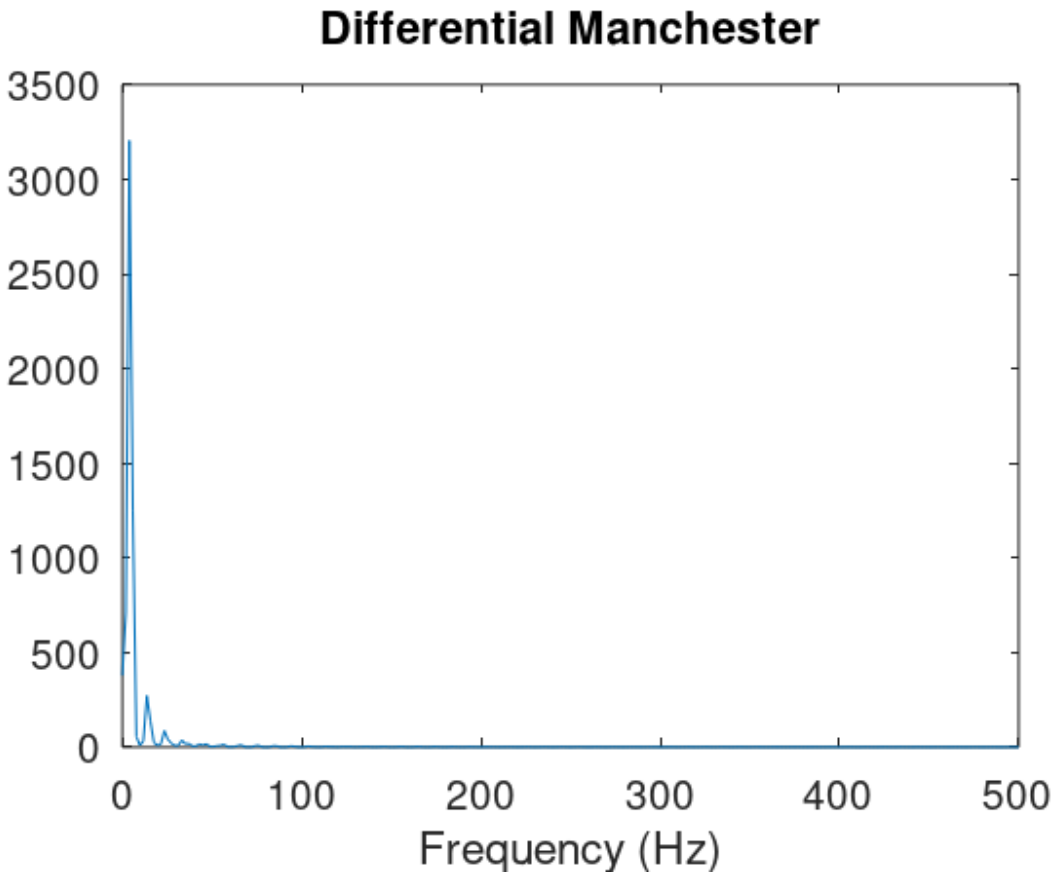

Рисунок 3.29: Дифференциальное манчестерское кодирование: спектр сигнала

4 Выводы

В ходе данной лабораторной работы я изучил методы кодирования и модуляции сигналов с помощью высокоуровнего языка программирования Octave. Научился определять спектра и параметров сигнала. Были продемонстрированы принципы модуляции сигнала на примере аналоговой амплитудной модуляции. Так же были исследованы свойства самосинхронизации сигнала.