РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра теории вероятностей и кибербезопасности

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

дисциплина: Сетевые технологии

Студент: Бансимба Клодели Дьегра

Студ. билет № 1032215651

Группа: НПИбд-02-22

МОСКВА

2024 г.

1-Цель работы

Изучить методы кодирования и модуляции сигналов с помощью высокоуровнего языка программирования octave. Определить спектр и параметры сигнала. Продемонстрировать принципы модуляции сигнала на примере аналоговой амплитудной модуляции. Исследовать свойства самосинхронизации сигнала.

2- Задание

1. Построить графики в octave:
   - 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 экспортировать в файлы формата .eps, .png.
   - Построить график функции $y = cos(x) + \frac{1}{3}cos(3x) + \frac{1}{5}cos(5x)$ на интервале $[-10; 10]$, используя octave и функцию `plot`. График экспортировать в файлы формата .eps, png.

2. Разложить импульсный сигнал в частичный ряд Фурье:
   - Разработать код m-файла, результатом выполнения которого являются графики меандра, реализованные с различным количеством гармоник.
3. Определить спектр и параметры сигнала:
   - Определить спектр двух отдельных сигналов и их суммы;
   - Выполнить задание с другой частотой дискретизации. Пояснить, что будет, если взять частоту дискретизации меньше 80 Гц?
4. Продемонстрировать принципы модуляции сигнала на примере аналоговой амплитудной модуляции;
5. По заданным битовым последовательностям требуется получить кодиро-

ванные сигналы для нескольких кодов, проверить свойства самосинхронизуемости кодов,получить спектры.

1 Выполнение лабораторной работы

1.1 Задание №1

После запуска в моей ОС octave с оконным интерфейсом я перешел в окно редактора, создал новый сценарий и сохранил его в рабочий каталог с именем plot_sin.m. В окне редактора я представил следующий листинг по построению графика функции:

```
% 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");
```

После запуска сценария открывается окно с построенным графиком (рис. 3.1) и в рабочем каталоге появились файлы с графиками в форматах .eps, .png. (рис. 3.1.1)

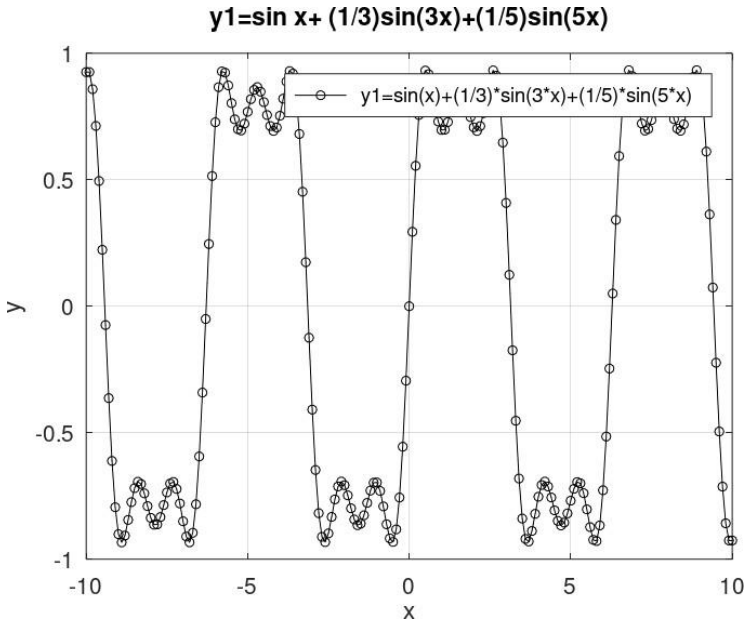


Рис. 3.1: График функции sin

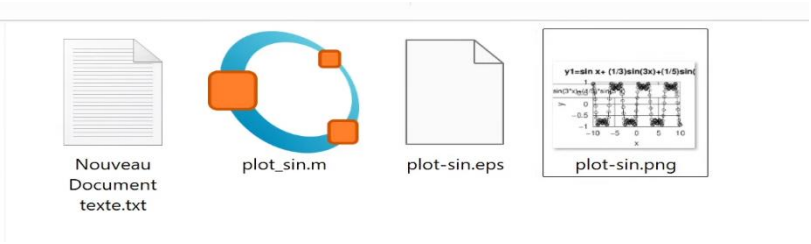


Рис. 3.1.1 : файлы с графиками в форматах .eps, .png

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

```
% Формирование массива x:
x=-10:0.1:10;
```

```
% Формирование массивов y1 и y2.
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);
% Построение первого графика функции:
plot(x,y1, "-ok; y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);","markersize",4)
hold on

% Построение второго графика функции:

plot(x,y2, "-; y2=cos(x)+(1/3)*cos(3*x)+(1/5)*cos(5*x);","markersize",4)
% Отображение сетки на графике
grid on;
% Подпись оси X:
xlabel('x');
% Подпись оси Y:
ylabel('y');
% Экспорт рисунка в файл .eps:
print ("plot-sin-cos.eps", "-mono", "-FArial:16", "-deps")
% Экспорт рисунка в файл .png:
print("plot-sin-cos.png");
```

После запуска сценария открывается окно с построенным графиком (рис. 3.2).

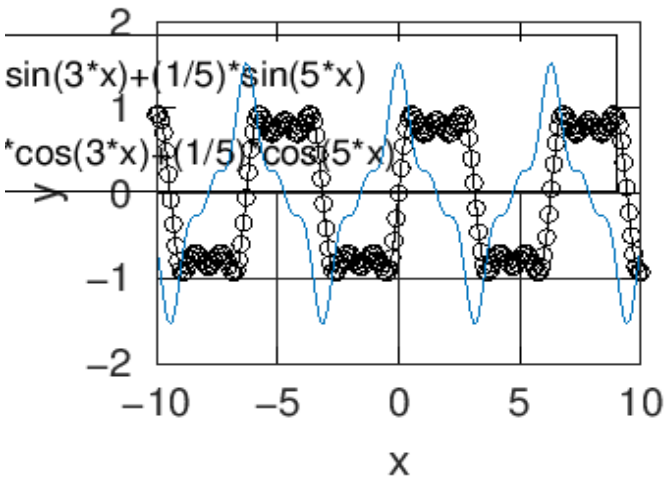


Рис. 3.2: График функций y1 и y2

1.2 Задание №2

Я создал новый сценарий и сохранила его в рабочий каталог с именем meandr.m. В коде созданного сценария задал начальные значения:

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

Разложение импульсного сигнал в форме меандра в частичный ряд Фурье можно задать формулой:

$$s(t) = \frac{A}{2} + \frac{2A}{\pi}cos(\frac{2\pi}{T}t) - \frac{1}{3}cos(s\frac{2\pi}{T}t) + \frac{1}{5}cos(5\frac{2\pi}{T}t) - \dots$$

или формулой:

$$s(t) = \frac{A}{2} + \frac{2A}{\pi}sin(\frac{2\pi}{T}t) - \frac{1}{3}sin(s\frac{2\pi}{T}t) + \frac{1}{5}sin(5\frac{2\pi}{T}t) - \dots$$

Гармоники, образующие меандр, имеют амплитуду, обратно пропорциональную номеру соответствующей гармоники в спектре, следовательно можно задать их так:

```
% 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
% Экспорт рисунка в файл .png:
print("plot-meandr-cos.png");
```

Далее экспортировала полученный график в файл в формате .png (рис. 3.3).

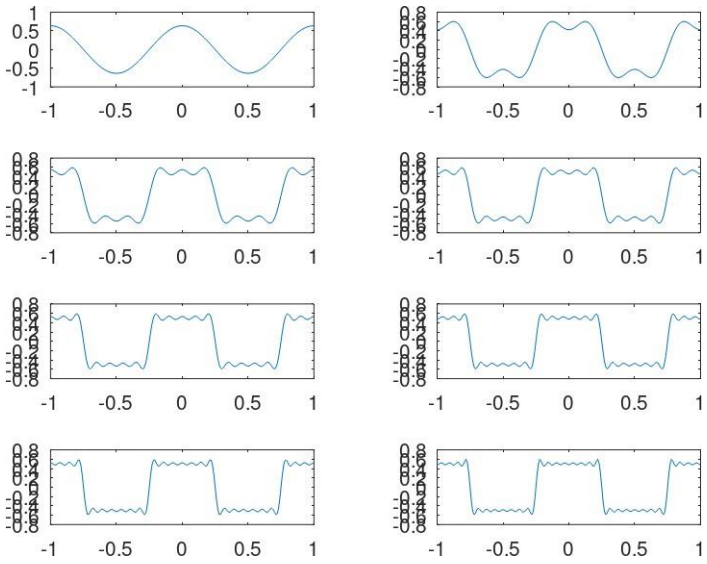


Рис. 3.3: Графики меандра, содержащего различное число гармоник

Скорректировал код для реализации меандра через синусы.

```
% 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
% Экспорт рисунка в файл .png:
print("plot-meandr-sin.png");
```

Получим соответствующие графики (рис. 3.4).

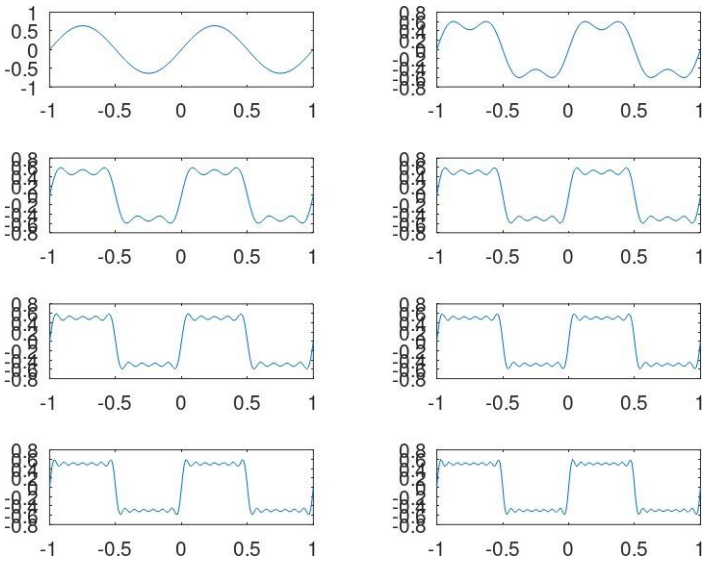


Рис. 3.4: Графики меандра, содержащего различное число гармоник

1.3 Задание №3

В рабочем каталоге создал каталог spectre1, а в нем сценарий с именем spectre.m. Написал в сценарии следующий код:

```
% 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);
```

Построил графики сигналов (рис. 3.5).

```
% 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';
```

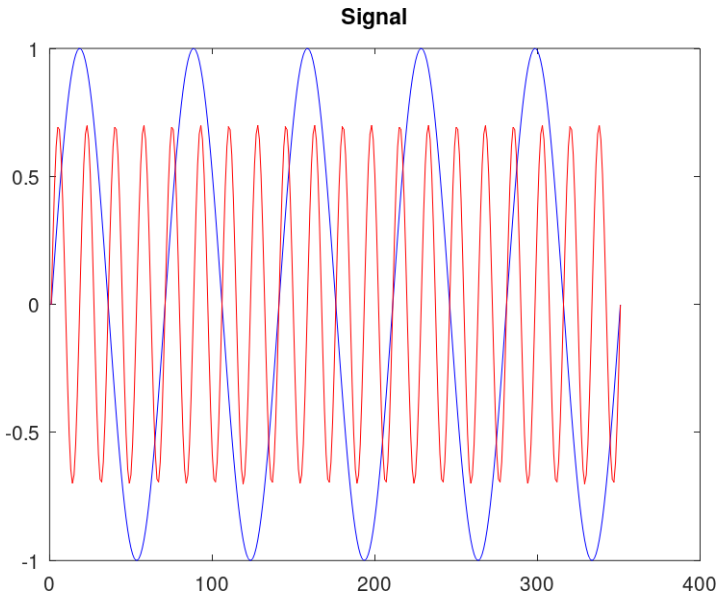


Рис. 3.5: Два синусоидальных сигнала разной частоты

Используем быстрое преобразование Фурье, чтобы найти спектры сигналов (рис. 3.6).

```
% 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';
```

Учитывая некоторые неточности преобразования Фурье, нужно скорректиро-

вать график спектра(рис. 3.7).

```
% 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';
```

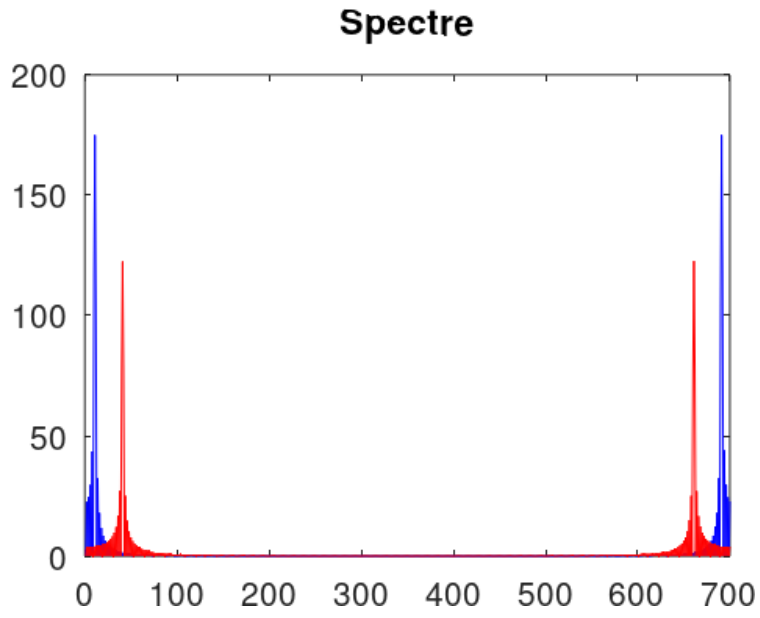


Рис. 3.6: График спектров синусоидальных сигналов

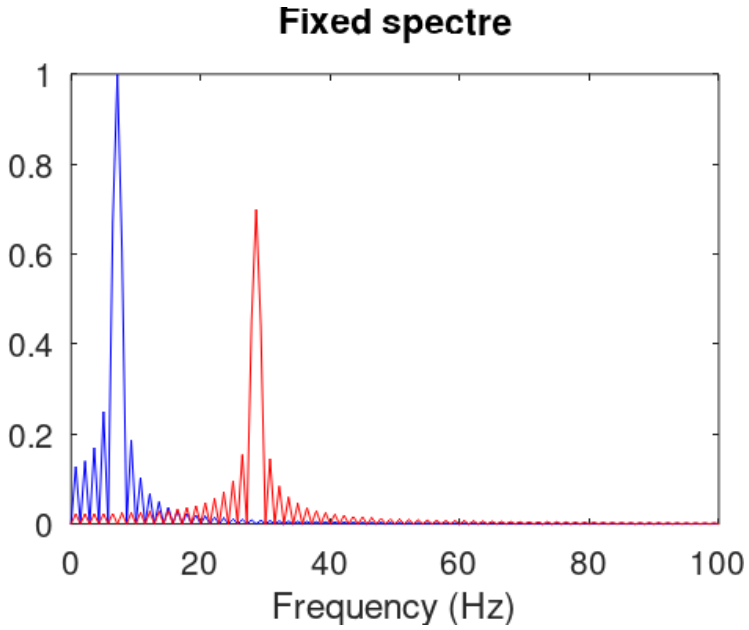


Рис. 3.7: Исправленный график спектров синусоидальных сигналов

Далее я нашел спектр суммы рассмотренных сигналов (рис. 3.8), создав каталог spectr_sum и файл в нём spectre_sum.m со следующим кодом:

```
% 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:
% 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';
```

В результате получился аналогичный предыдущему результат (рис. 3.9), т.е. спектр суммы сигналов должен быть равен сумме спектров сигналов, что вытекает из свойств преобразования Фурье.

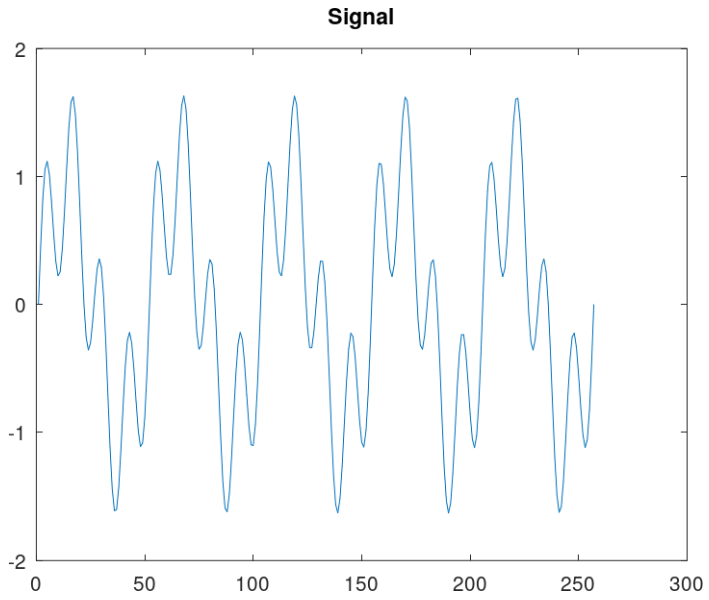


Рис. 3.8: Суммарный сигнал

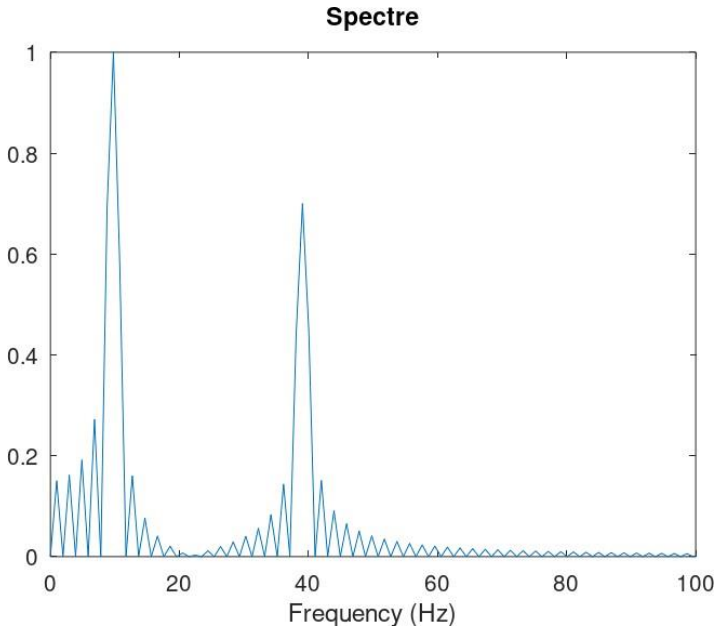


Рис. 3.9: Спектр суммарного сигнала

Также нас просят выполнить задание с другой частотой дискретизации. Я попробовал взять частоту меньше 512, например, 300, и больше 512, например 700.

И получилось, что чем меньше частоту дискретизации мы берем — тем меньше ступеней и больше шумов квантования получаем (на графиках получаются более ломанные линии). И, соответственно, повышение частоты дискретизации позволяет описать волну точнее и уменьшить шумы (линии идут плавнее).

Что же будет, если взять часоту дискретизации меньше 80 Гц? Как глаголит теорема Котельникова: если аналоговый сигнал $x(t)$ имеет конечный (ограниченный по ширине) спектр, то он может быть восстановлен однозначно и без потерь по своим отсчётам (гармоникам), взятым с частотой, строго большей удвоенной верхней частоты f_c : $f > 2f_c$. А поскольку мы задали такие начальные значения, что частота второго сигнала (Гц): `f2 = 40`, то получаем, что частоту дискретизации надо брать больше 80 Гц для точного воспроизведения сигнала. Чтобы наглядно увидеть доказательство теоремы, я изменил в коде частоту дискретизации на 50 Гц: `fd = 50;`

И получила “плохие” графики (рис. 3.10-3.14):

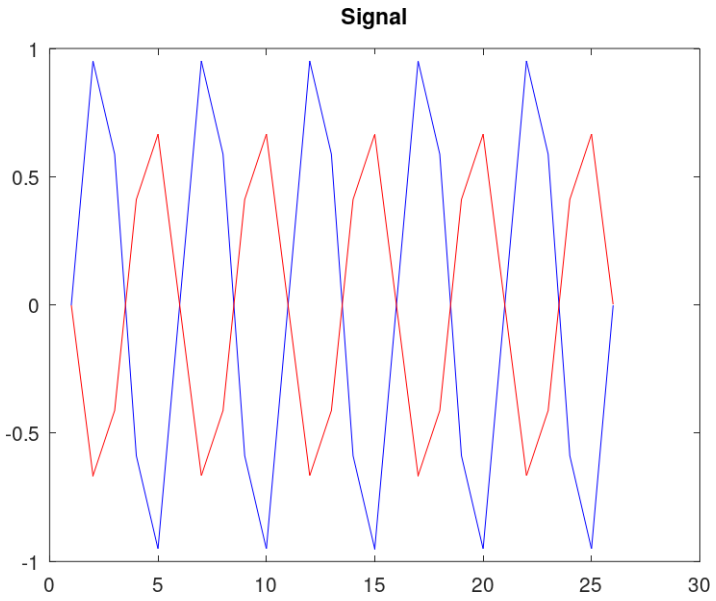


Рис. 3.10: Два синусоидальных сигнала разной частоты

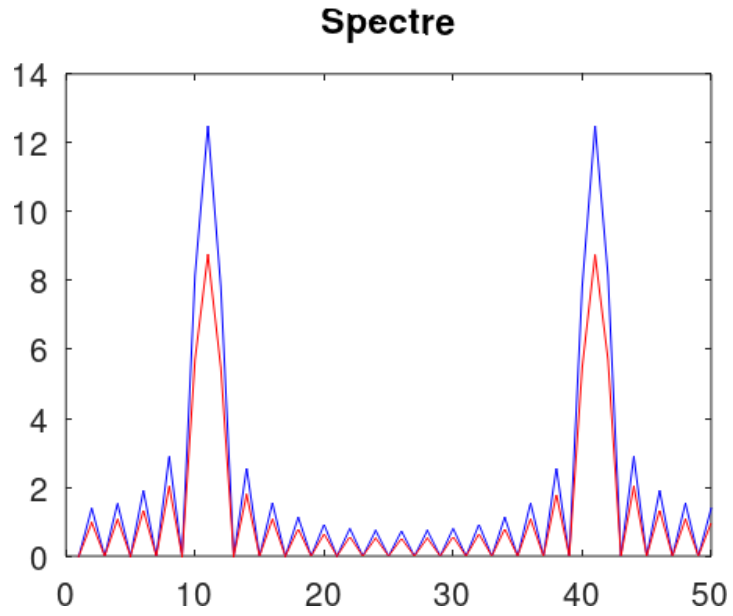


Рис. 3.11: График спектров синусоидальных сигналов

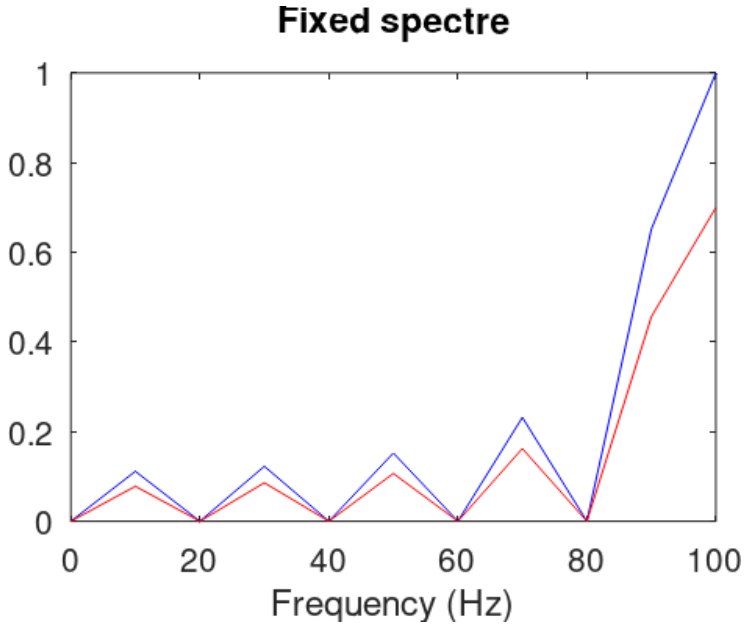


Рис. 3.12: Исправленный график спектров синусоидальных сигналов

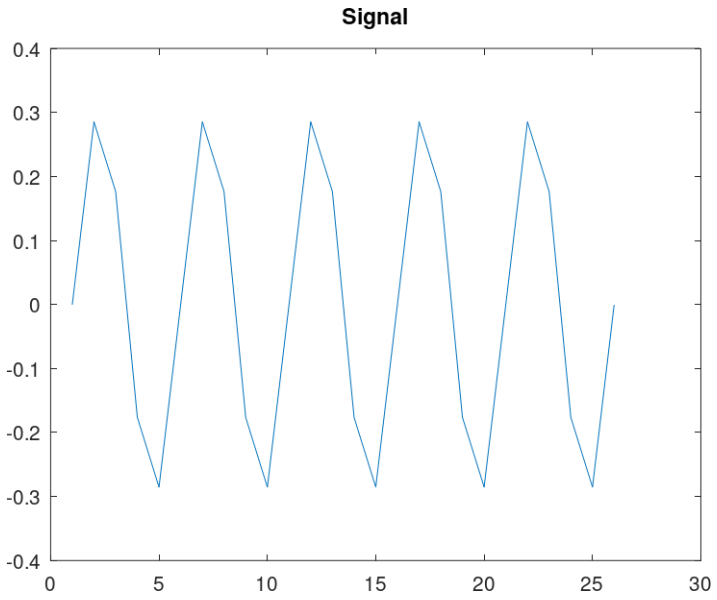


Рис. 3.13: Суммарный сигнал

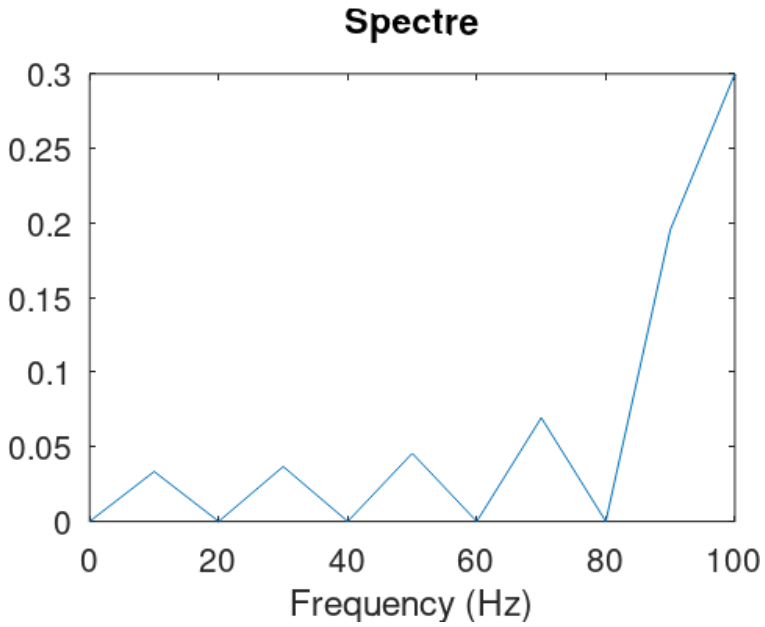


Рис. 3.14: Спектр суммарного сигнала

1.4 Задание №4

Продемонстрируем аналоговую амплитудную модуляцию (рис. 3.15). В рабочем каталоге создал каталог modulation, а в нем файл с именем am.m. В этом файле написал следующий код:

```
% modulation/am.m
% Создание каталогов signal и spectre для размещения графиков:
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 = 512;
% 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';
```

Далее построим спектр произведения, который представляет собой свертку спекторв (рис. 3.16).

```
% 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';
```

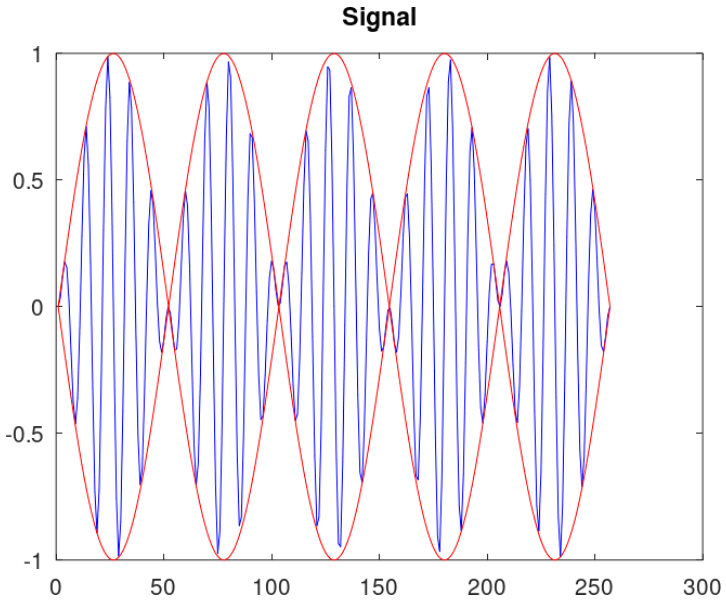


Рис. 3.15: Сигнал и огибающая при амплитудной модуляции

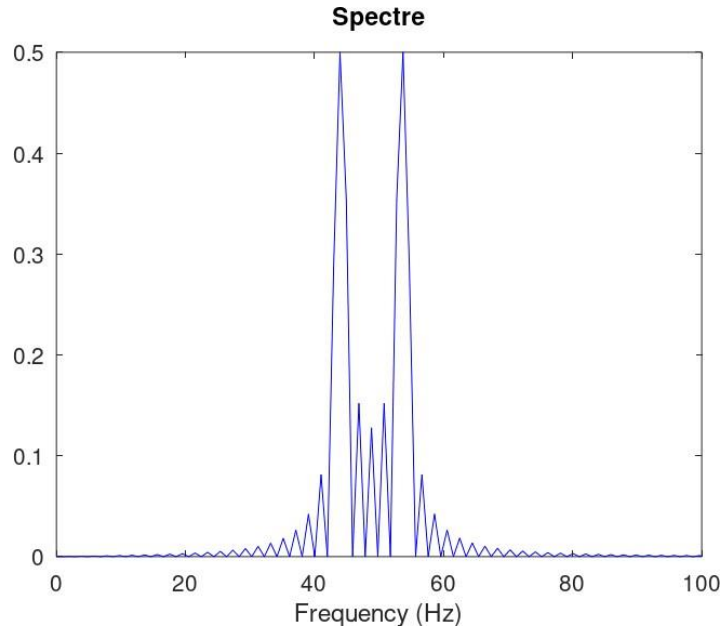


Рис. 3.16: Спектр сигнала при амплитудной модуляции

1.5 Задание №5

В рабочем каталоге создал каталог coding, а в нем файлы main.m, maptowave.m,unipolar.m,ami.m,bipolarnrz.m bipolarrz.m,manchester.m, diffmanc.m, calcspectre.m. Проверила, что установлен пакет расширений signal.

В файле main.m подключил пакет signal и задал входные кодовые последовательности, а затем прописал вызовы функций для построения графиков модуляций кодированных сигналов для заданной кодовой 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 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");
% 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';
```

В файле maptowave.m прописал 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 прописал соответствующие функции преобразования кодовой последовательности data с вызовом функции maptowave для построения соответствующего графика.

Униполярное кодирование:

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

Кодирование AMI:

```
% 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);
```

Кодирование NRZ:

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

Кодирование RZ:

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

Манчестерское кодирование:

```
% 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);
```

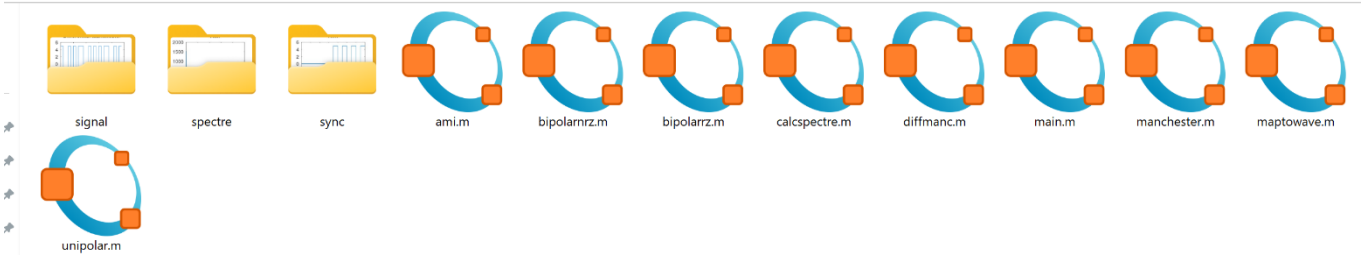
Дифференциальное манчестерское кодирование:

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

В файле calcspectre.m прописала функцию построения спектра сигнала:

```
% calcspectre.m
% Функция построения спектра сигнала:
function spectre = calcspectre(wave)
```

```
  % 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:Fd3));
xlabel('Frequency (Hz)');
```



После запуска главного скрипта main.m в каталоге signal получились файлы с графиками кодированного сигнала (рис. 3.17-3.22), в каталоге sync — файлы с графиками, иллюстрирующими свойства самосинхронизации (рис. 3.23-3.28), в каталоге spectre — файлы с графиками спектров сигналов (рис. 3.29-3.34).

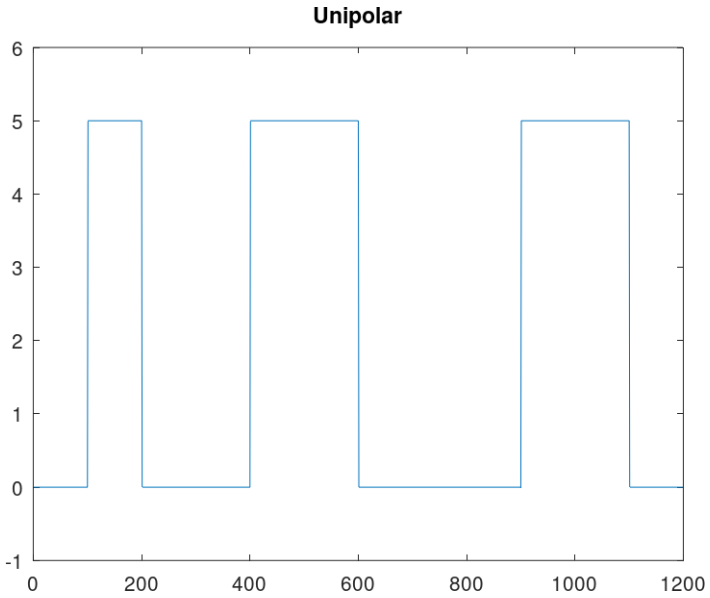


Рис. 3.17: Униполярное кодирование

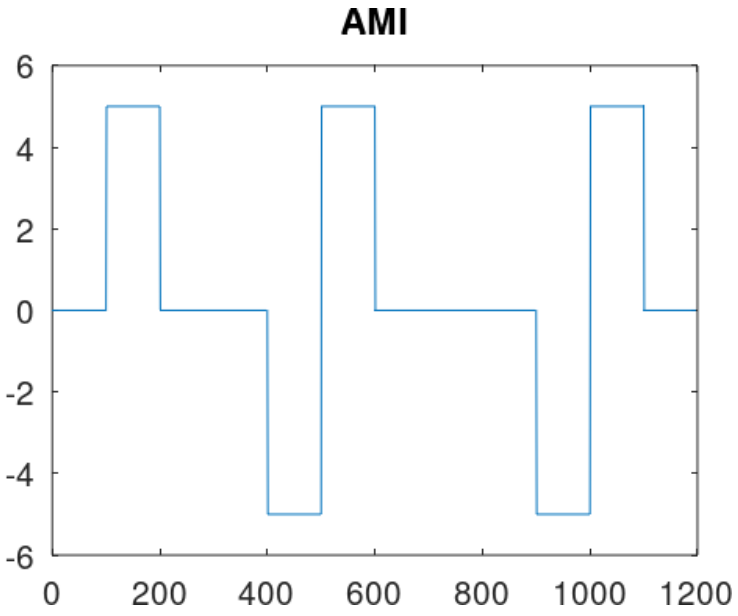


Рис. 3.18: Кодирование AMI

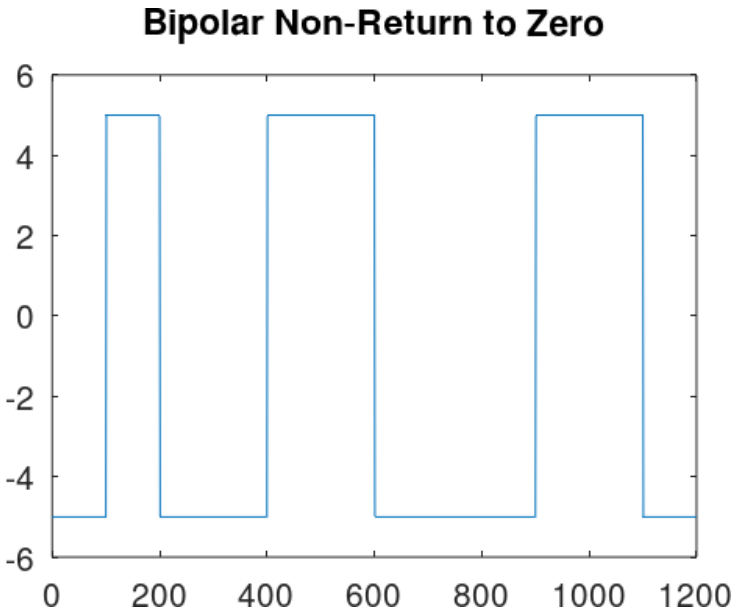


Рис. 3.19: Кодирование NRZ

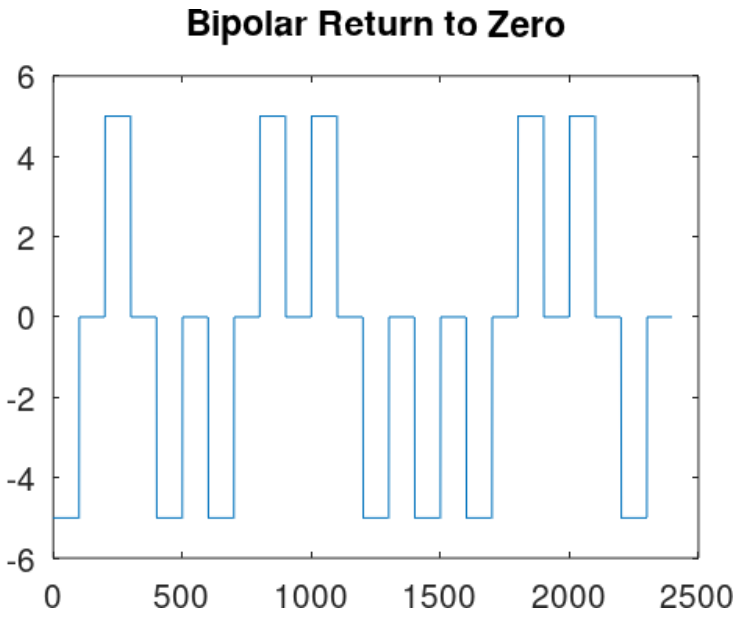


Рис. 3.20: Кодирование RZ

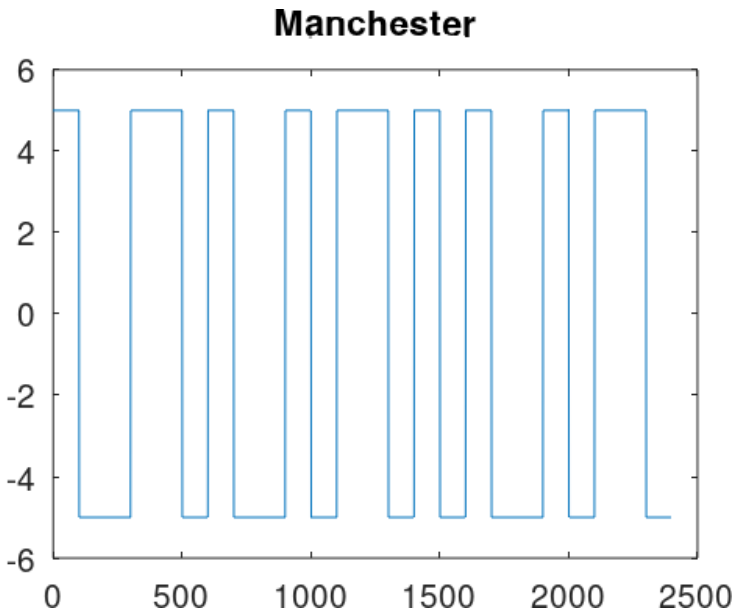


Рис. 3.21: Манчестерское кодирование

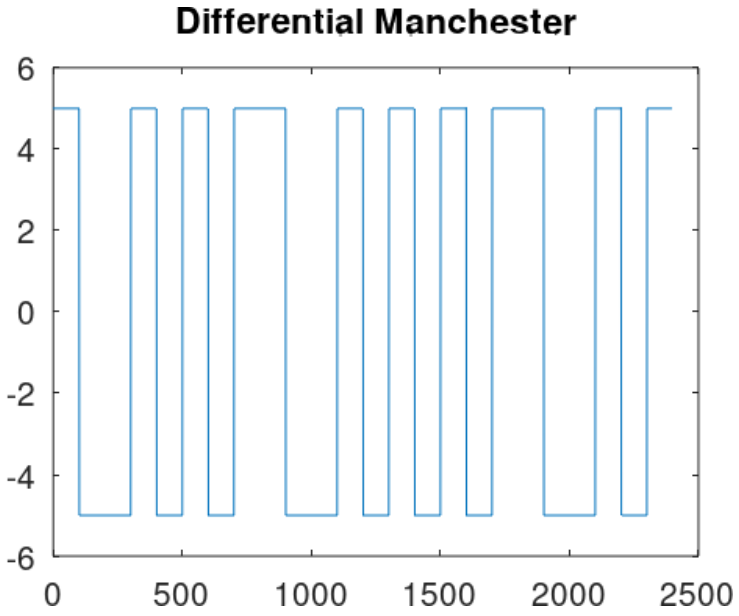


Рис. 3.22: Дифференциальное манчестерское кодирование

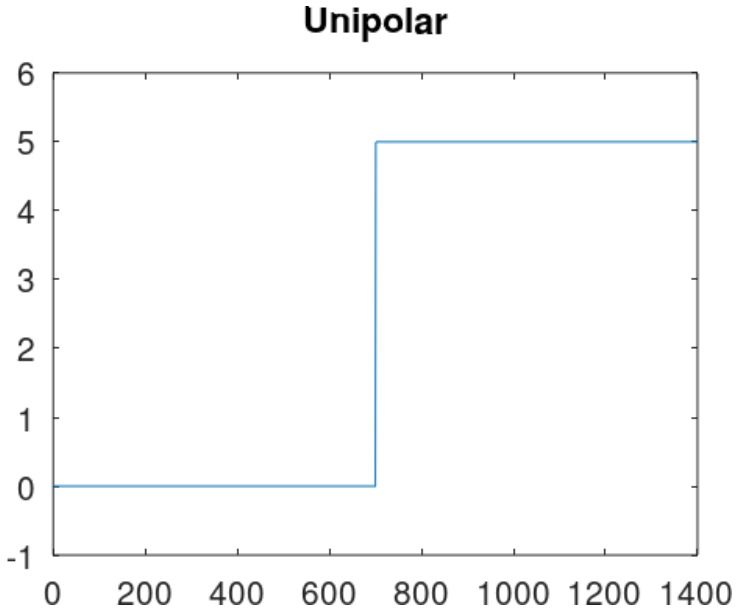


Рис. 3.23: Униполярное кодирование: нет самосинхронизации

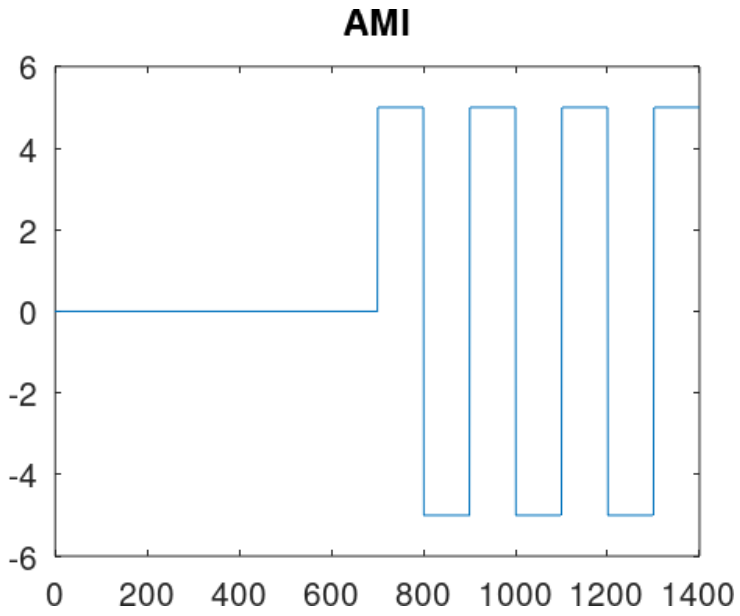


Рис. 3.24: Кодирование AMI: самосинхронизация при наличии сигнала

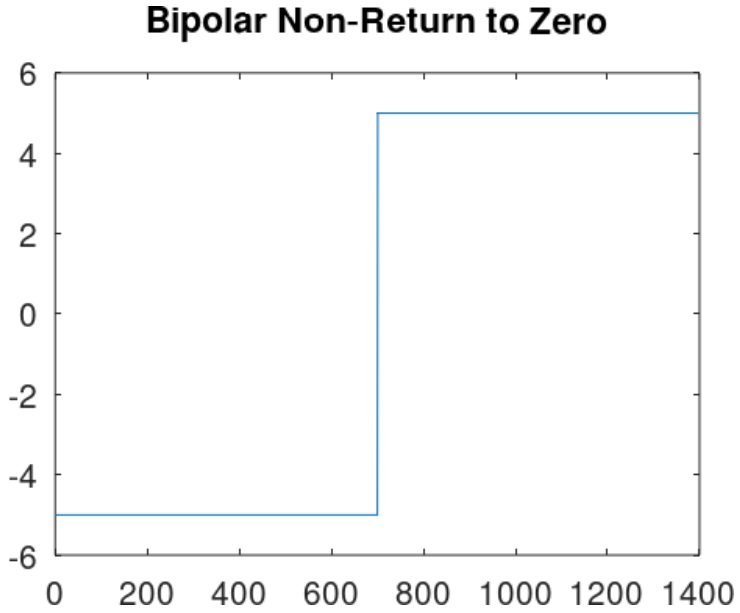


Рис. 3.25: Кодирование NRZ: нет самосинхронизации

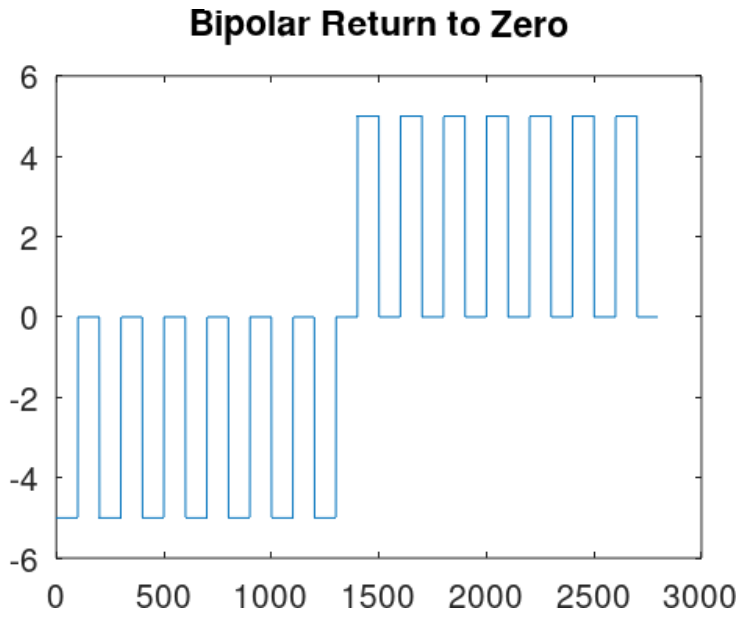


Рис. 3.26: Кодирование RZ: есть самосинхронизация

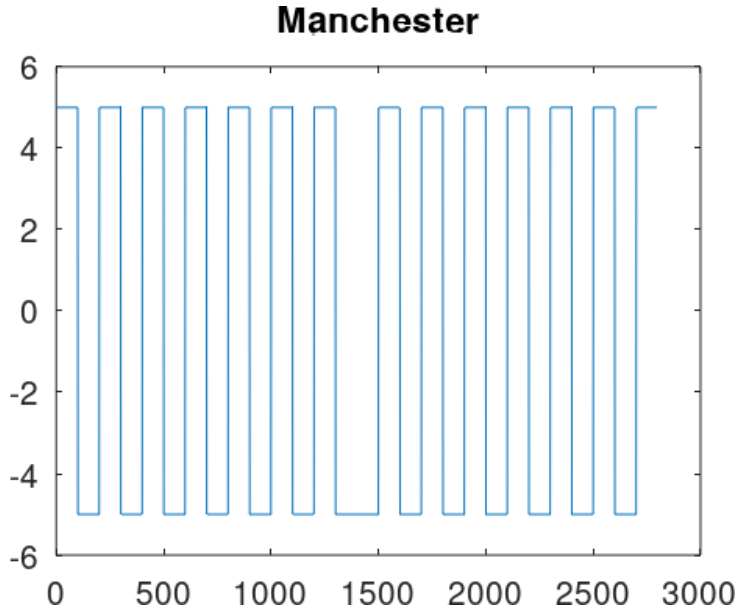


Рис. 3.27: Манчестерское кодирование: есть самосинхронизация

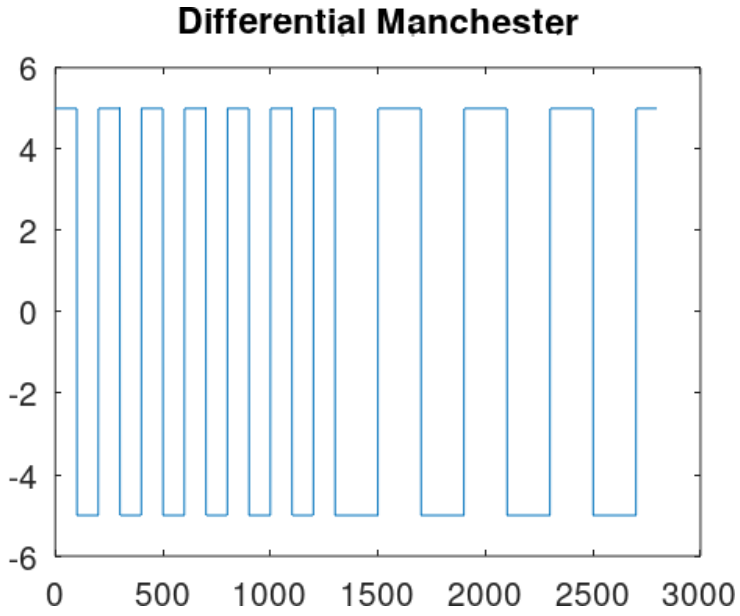


Рис. 3.28: Дифференциальное манчестерское кодирование: есть самосинхронизация

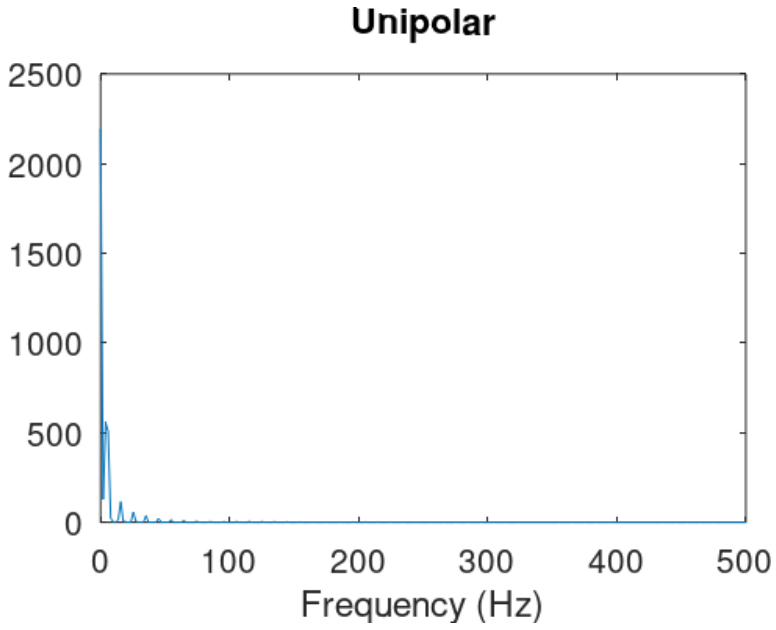


Рис. 3.29: Униполярное кодирование: спектр сигнала

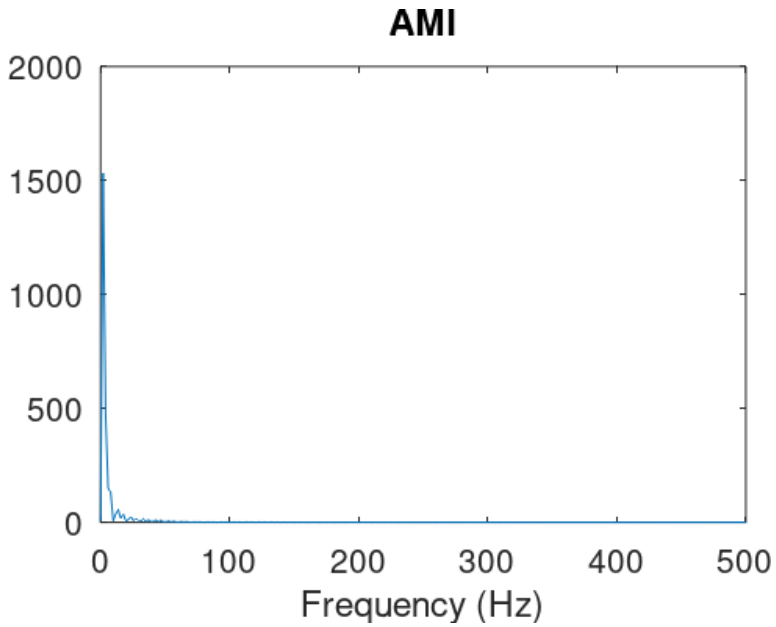


Рис. 3.30: Кодирование AMI: спектр сигнала

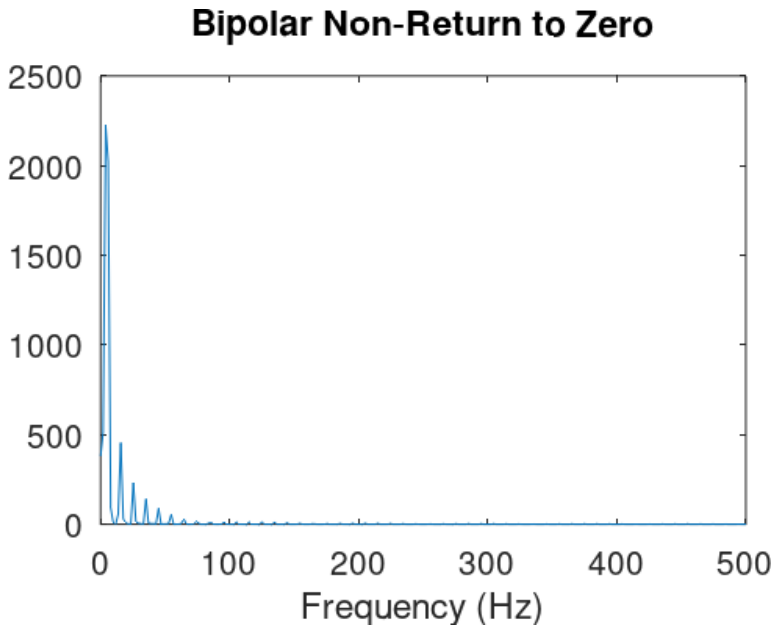


Рис. 3.31: Кодирование NRZ: спектр сигнала

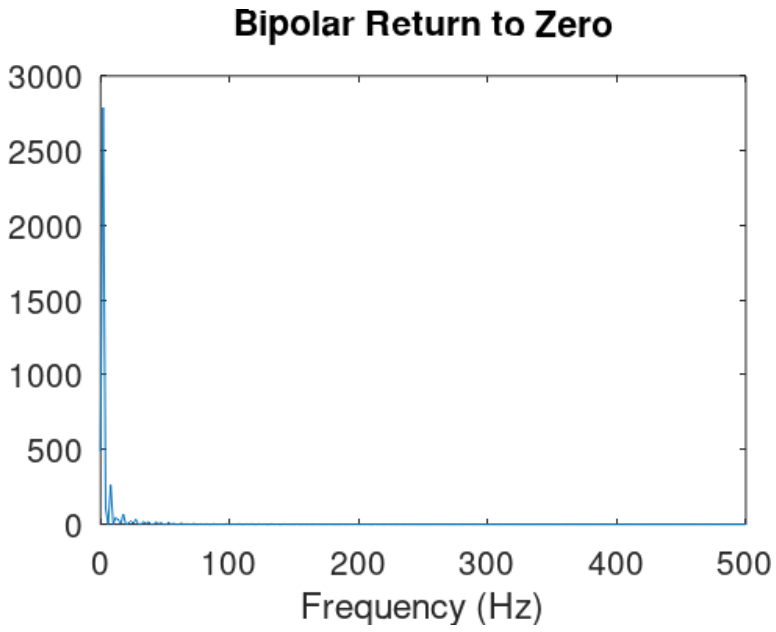


Рис. 3.32: Кодирование RZ: спектр сигнала

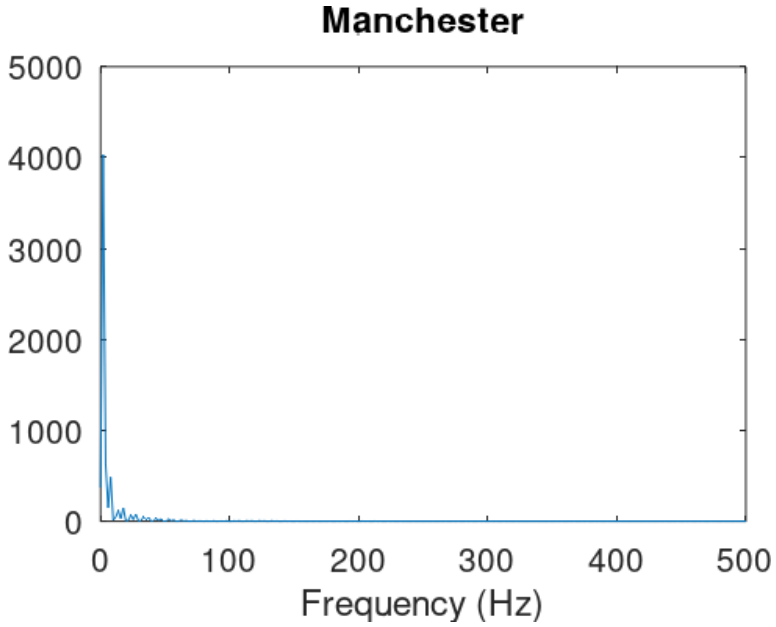


Рис. 3.33: Манчестерское кодирование: спектр сигнала

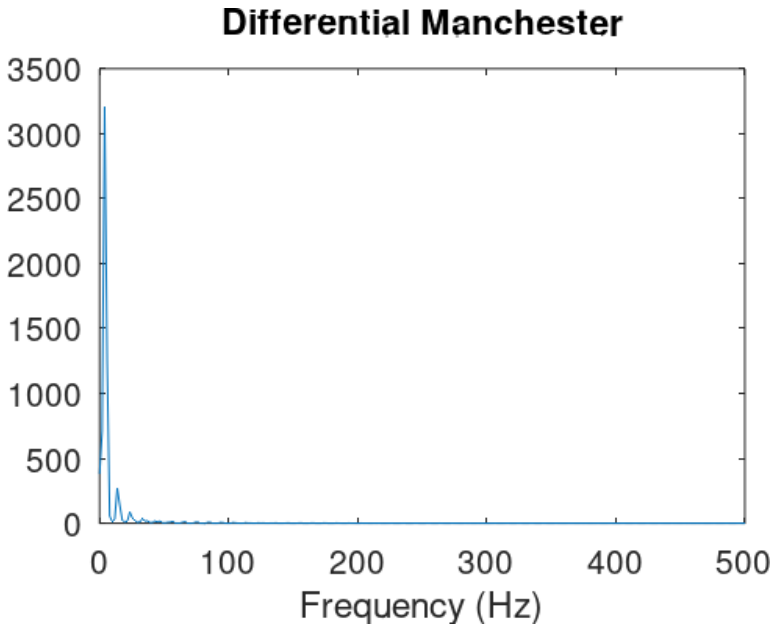


Рис. 3.34: Дифференциальное манчестерское кодирование: спектр сигнала

2 Выводы

В процессе выполнения данной лабораторной работы я изучил методы кодирования и модуляции сигналов с помощью высокоуровнего языка программирования octave.

Определил спектр и параметры сигнала. Показал принципы модуляции сигнала на примере аналоговой амплитудной модуляции. А также исследовала свойства самосинхронизации сигнала.