

Лабораторная работа №1

Сетевые технологии

Мурашов Иван Вячеславович

2025-09-13

Содержание I

1 Цель работы

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.

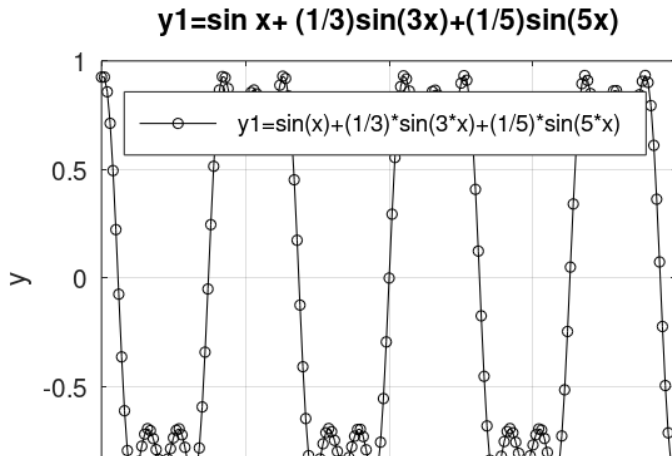
2 Построение графиков в Octave

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

```
% Очистка экрана и установка оси X:
x=-10:0.1:10;
% Очистка экрана и установка оси Y.
y1=sin(x)+1/3*sin(3*x)+1/5*sin(5*x);
% Установка формата вывода:
plot(x,y1, "-ok; y1=sin(x)+
(1/3)*sin(3*x)+(1/5)*sin(5*x);", "markersize", 4)
% Установка формата вывода
grid on;
% Установка оси X:
xlabel('x');
```

3 Построение графиков в Octave

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



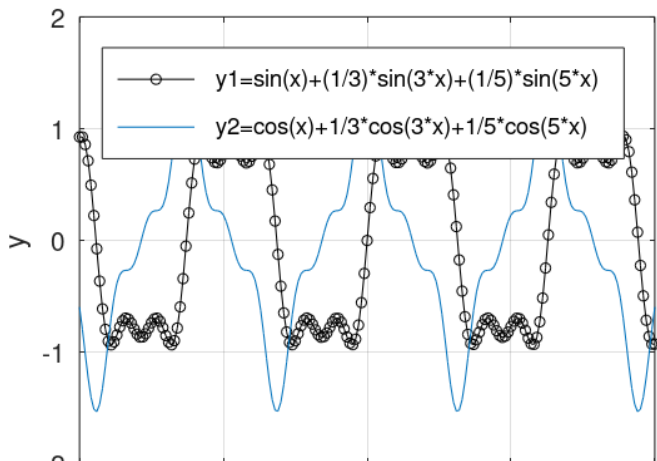
4 Построение графиков в Octave

Сохраняю сценарий под другим названием и измените его так, чтобы на одном графике располагались отличающиеся по типу линий графики функций $y_1 = \sin x + 1/3 * \sin 3x + 1/5 * \sin 5x$ и $y_2 = \cos x + 1/3 * \cos 3x + 1/5 * \cos 5x$. 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);
% #####:
plot(x,y1, "-ok; y1=sin(x)+(1/3)*sin(3*x)+(1/5)*sin(5*x);", "ma
hold on
plot(x,y2, "-; y2=cos(x)+1/3*cos(3*x)+1/5*cos(5*x);", "markersi
% #####
grid on;
```

5 Построение графиков в Octave

Имеем график (рис. [-@fig:002]).



6 Разложение импульсного сигнала в частичный ряд Фурье

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

```
% meandr.m  
% [1x8 double] [1x8 double] ([1x8 double]):  
N=8;  
% [1x8 double] [1x16 double]:  
t=-1:0.01:1;  
% [1x8 double] [1x8 double]:  
A=1;  
% [1x8 double]:  
T=1;
```


7 Разложение импульсного сигнала в частичный ряд Фурье

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

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





8 Разложение импульсного сигнала в частичный ряд Фурье

Далее задаём массив значений гармоник массив элементов ряда:

```
%  $n_{h'}$   $n_{h'}$ :
harmonics=cos(2 * pi *  $n_{h'}$  * t/T);
%  $n_{h'}$   $n_{h'}$   $n_{h'}$ :
s1=harmonics.* repmat(Am',1,length(t));
```

9 Разложение импульсного сигнала в частичный ряд Фурье

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:

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

10 Разложение импульсного сигнала в частичный ряд Фурье

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

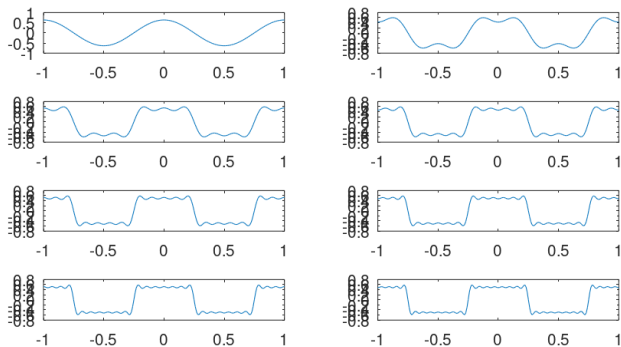


Рисунок 3: Графики меандра, содержащего различное число гармоник

11 Разложение импульсного сигнала в частичный ряд Фурье

Корректируем код для реализации меандра через синусы:

```
% meandr.m
% figure(1) hold on; % (figure(1)) :
N=8;
% figure(2) hold on; :
t=-1:0.01:1;
% figure(3) hold on; :
A=1;
% figure(4):
T=1;
% figure(5) hold on;
nh=(1:N)*2-1;
% figure(6) hold on; nh, figure(7) hold on; sin:
Am=2/pi ./ nh;
% Am(2:2:end) = Am(2:2:end); % not needed
```

12 Разложение импульсного сигнала в частичный ряд Фурье

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

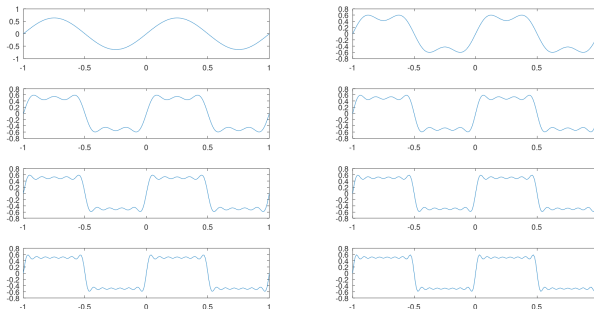


Рисунок 4: Графики меандра, содержащего различное число гармоник

13 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 signal spectre
% :
mkdir 'signal';
mkdir 'spectre';
% ( ):
tmax = 0.5;
% ( ) ( ):
fd = 512;
% ( ):
f1 = 10;
% ( ):
f2 = 40;
```

14 Определение спектра и параметров сигнала

Далее в коде задаю два синусоидальных сигнала разной частоты:

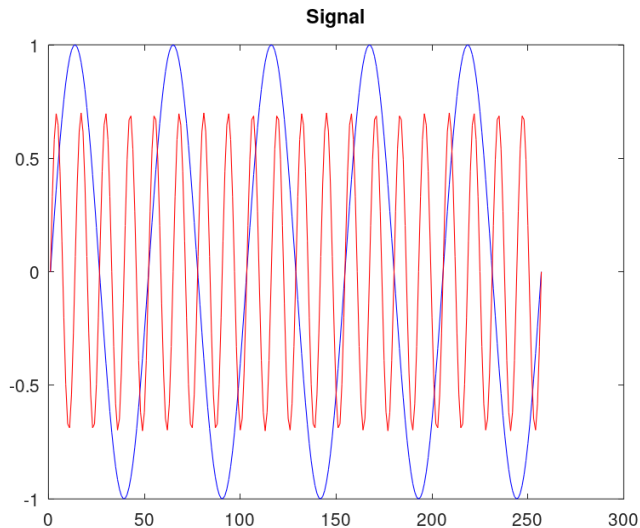
```
% Создание вектора времени t Создание вектора частот f Создание вектора амплитуд a Создание вектора фаз phi:  
signal1 = a1*sin(2*pi*t*f1);  
signal2 = a2*sin(2*pi*t*f2);
```


15 Определение спектра и параметров сигнала

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

```
% 1-й сигнал:  
plot(signal1, 'b');  
% 2-й сигнал:  
hold on  
plot(signal2, 'r');  
hold off  
title('Signal');  
% Сохраняю график сигнала:  
print 'signal/spectre.png';
```

16 Определение спектра и параметров сигнала



17 Определение спектра и параметров сигнала

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

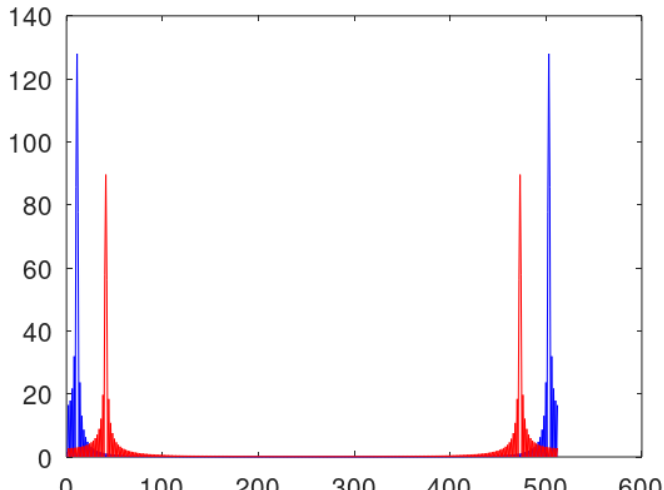
```
%
%
%
spectre1 = abs(fft(signal1,fd));
%
spectre2 = abs(fft(signal2,fd));
%
plot(spectre1,'b');
hold on
plot(spectre2,'r');
hold off
title('Spectre');
print 'spectre/spectre.png';
```

18 Определение спектра и параметров сигнала

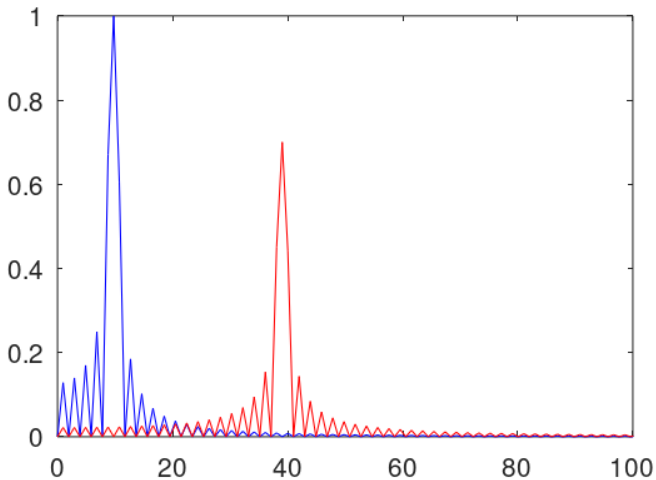
Учитывая реализацию преобразования Фурье, корректирую график спектра (рис. [-@fig:007]): отбрасываем дублирующие отрицательные частоты, а также принимаем в расчёт то, что на каждом шаге вычисления быстрого преобразования Фурье происходит суммирование амплитуд сигналов. Для этого добавляю в файл `spectre.m`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

```
%
%
%
%
%
f = 1000*(0:fd2)./(2*fd);
%
spectre1 = 2*spectre1/fd2;
spectre2 = 2*spectre2/fd2;
%
plot(f,spectre1(1:fd2+1),'b');
hold on
plot(f,spectre2(1:fd2+1),'r');
```

19 Определение спектра и параметров сигнала

Spectre

20 Определение спектра и параметров сигнала

Fixed spectre

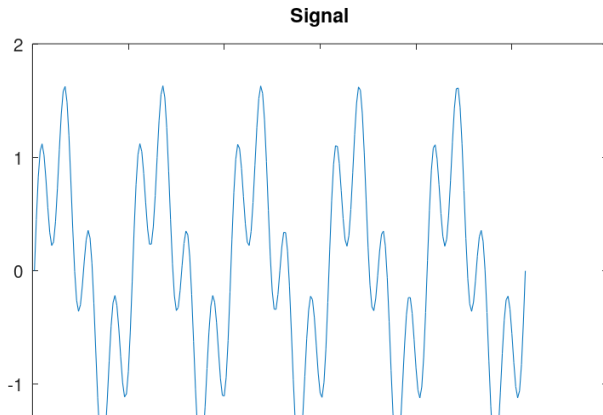
21 Определение спектра и параметров сигнала

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

```
ctr_sum = zeros(1, 1000) spectre_sum.m = zeros(1, 1000):
% spectr_sum/spectre_sum.m
% 1000x1000 signal = spectre 1000 1000x1000
% 1000x1000:
mkdir 'signal';
mkdir 'spectre';
% 1000x1000 (1):
tmax = 0.5;
% 1000x1000 1000x1000 (1) (1000x1000 1000x1000):
fd = 512;
% 1000x1000 1000x1000 1000x1000 (1):
f1 = 10;
% 1000x1000 1000x1000 1000x1000 (1):
```

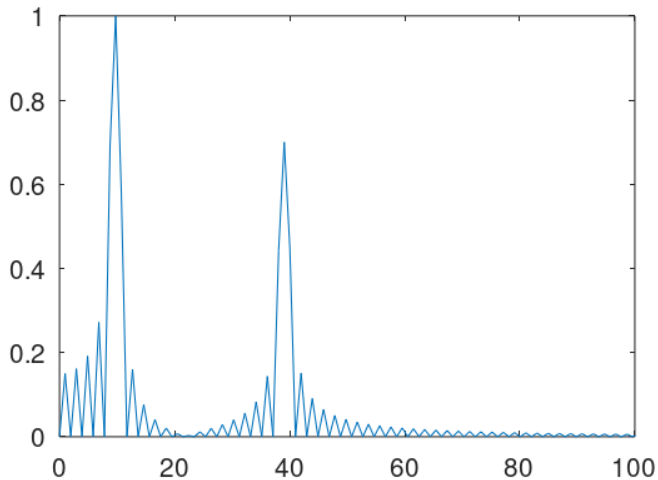
22 Определение спектра и параметров сигнала

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



23 Определение спектра и параметров сигнала

Spectre



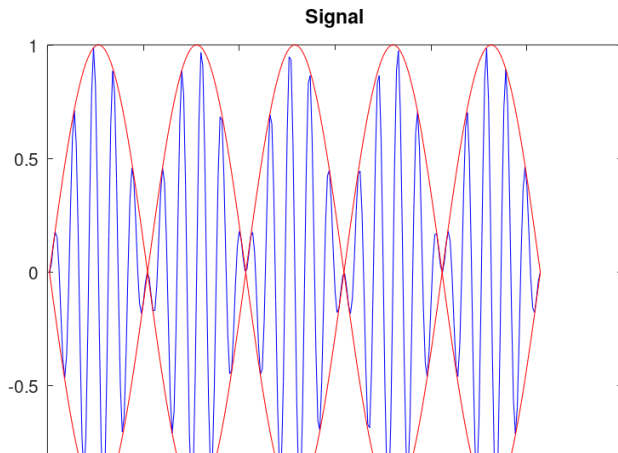
24 Амплитудная модуляция

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

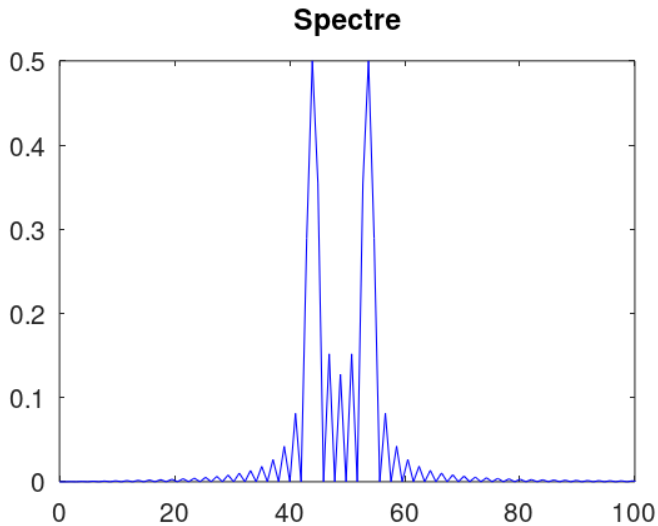
```
% modulation/am.m
% ##### signal & spectre #####
% #####:
mkdir 'signal';
mkdir 'spectre';
% ##### & ##### 50 & 5
% ##### ( )
tmax = 0.5;
% ##### ( ) ( )
fd = 512;
% ##### ( )
f1 = 5;
% ##### ( )
```

25 Амплитудная модуляция

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



26 Амплитудная модуляция



27 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

Создаю каталог coding и в нём файлы main.m, maptowave.m, unipolar.m, ami.m, bipolarnrz.m, bipolarrrz.m, manchester.m, diffmanc.m, calcspectre.m.

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

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

% [XXXXXXX XXXXXXX XXXXXXXXXXXXXXXXXXXXXXXX]:
data=[0 1 0 0 1 1 0 0 0 1 1 0];
% [XXXXXXX XXXXXXX XXXXXXXXXXXXXXXXXXXXXXXX XXX XXXXXXXXXXXXX
X XXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXX]:
data_sync=[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1];
```

28 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

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

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

29 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

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

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

30 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

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

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


31 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

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);
```

32 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

В файлах unipolar.m, ami.m, bipolarnrz.m, bipolarrrz.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
% ██████████ ██████████:
function wave=unipolar(data)
wave=maptowave(data);
```

Кодирование AMI:

```
██████████ AMI:
% coding/ami.m
% ██████████ AMI:
```

33 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

Кодирование NRZ:

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

Кодирование RZ:

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

34 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

Манчестерское кодирование:

```
% coding/manchester.m
% [XXXXXXXXXXXXXXXX] [XXXXXXXXXXXX]:
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
% [XXXXXXXXXXXXXXXXXXXX] [XXXXXXXXXXXXXXXXXXXX] [XXXXXXXXXXXXXXXXXXXX]
function wave=diffmanc(data)
data=filter(1,[1 1],data);
```

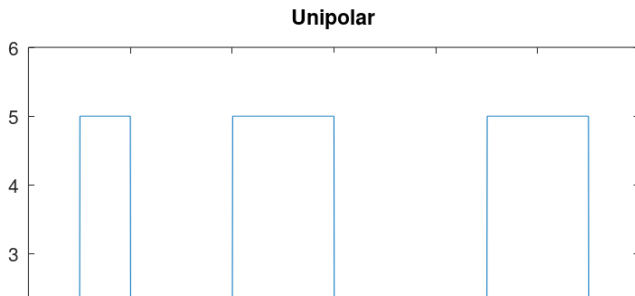
35 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

В файле calcspectre.m прописываем функцию построения спектра сигнала:

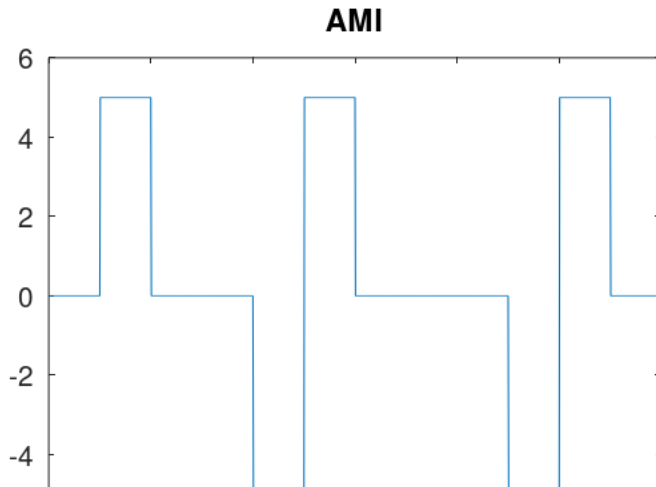
```
% calcspectre.m
% [spectre, f] = calcspectre(wave):
function [spectre, f] = calcspectre(wave)
% [spectre, f] = calcspectre(wave) (f):
Fd = 512;
Fd2 = Fd/2;
Fd3 = Fd/2 + 1;
X = fft(wave, Fd);
spectre = X.*conj(X)/Fd;
f = 1000*(0:Fd2)/Fd;
plot(f, spectre(1:Fd3));
xlabel('Frequency (Hz)');
```

36 Кодирование сигнала. Исследование свойства самосинхронизации сигнала

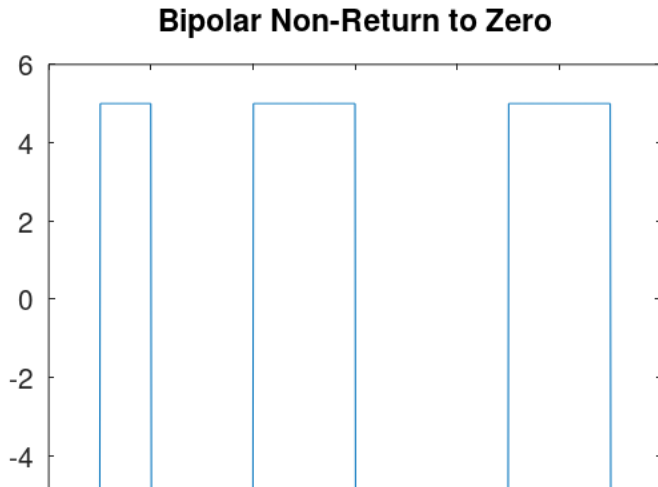
Запускаю главный скрипт `main.m`. В каталоге `signal` должны быть получены файлы с графиками кодированного сигнала (рис. [-@fig:012] – рис. [-@fig:017]), в каталоге `sync` — файлы с графиками, иллюстрирующими свойства самосинхронизации (рис. [-@fig:018] – рис. [-@fig:023]), в каталоге `spectre` — файлы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



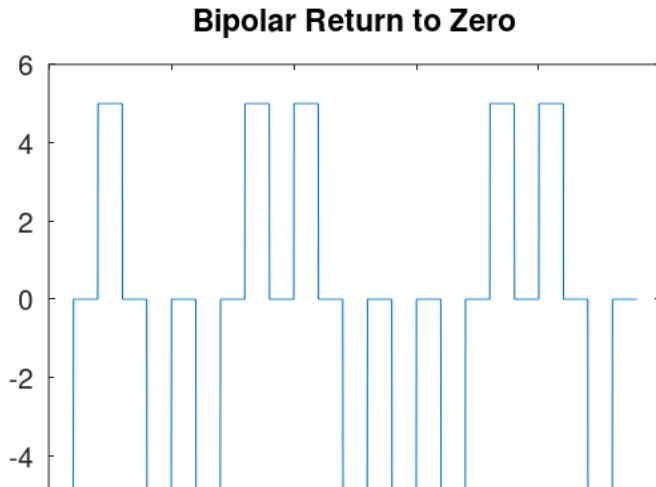
37 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



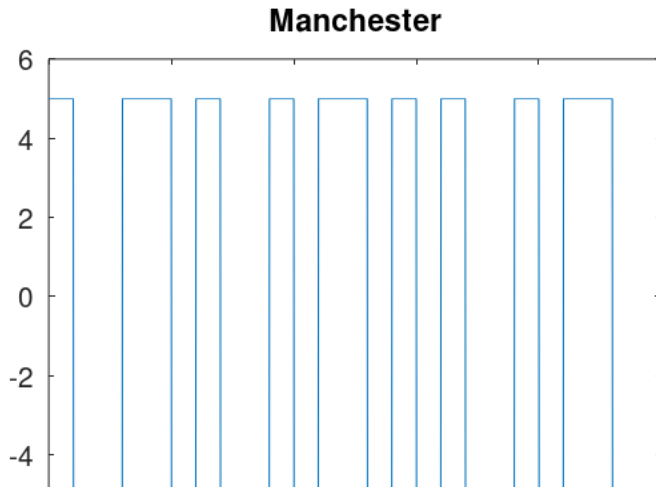
38 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



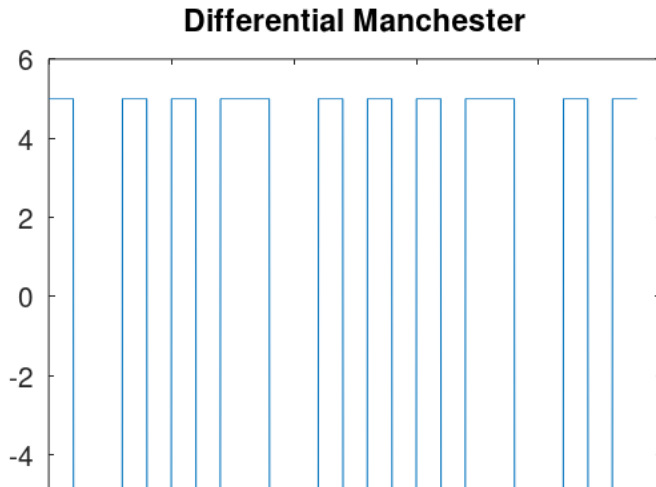
39 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



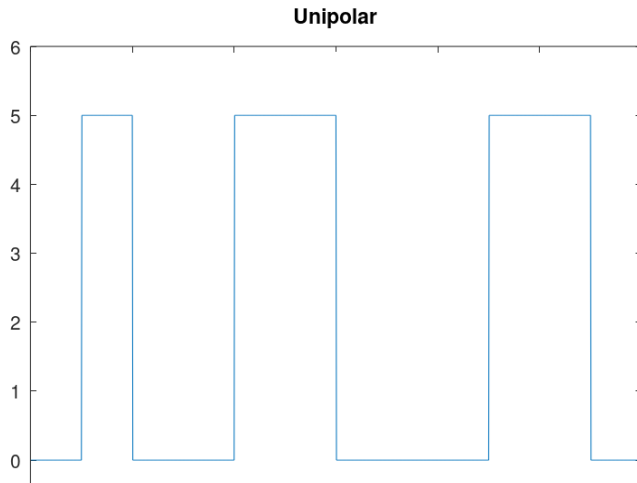
40 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



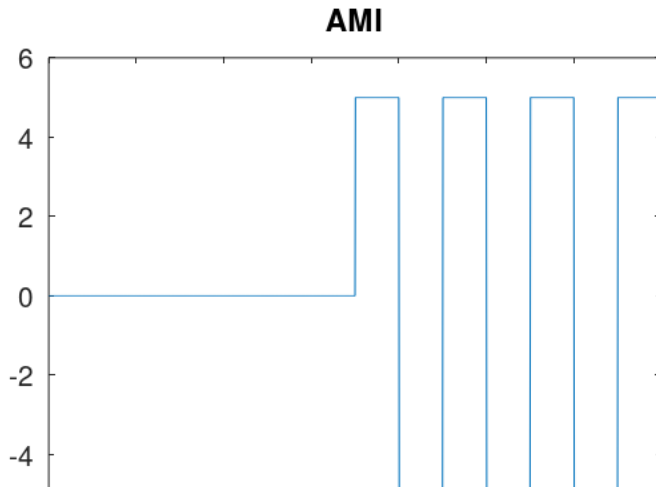
41 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



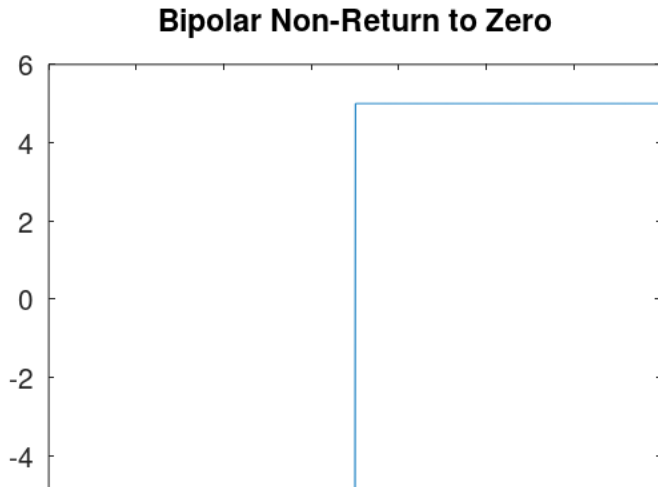
42 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



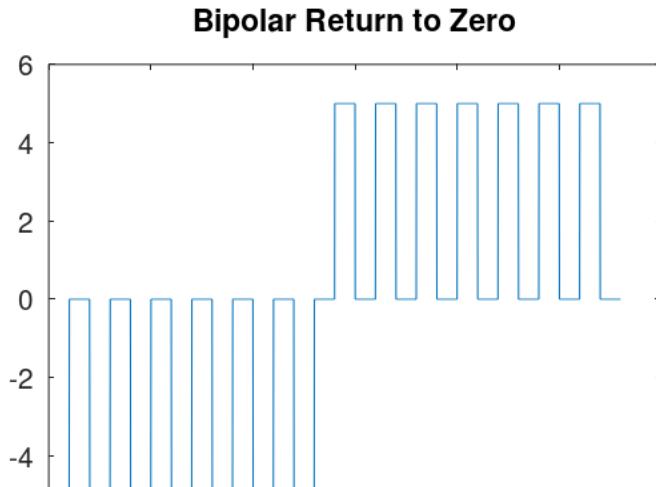
43 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



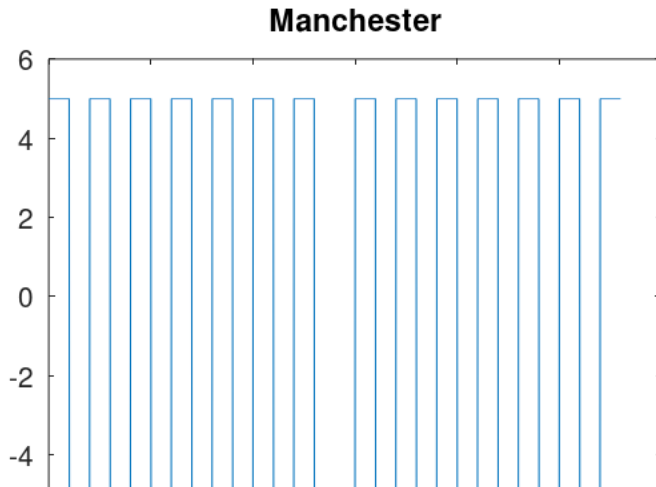
44 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



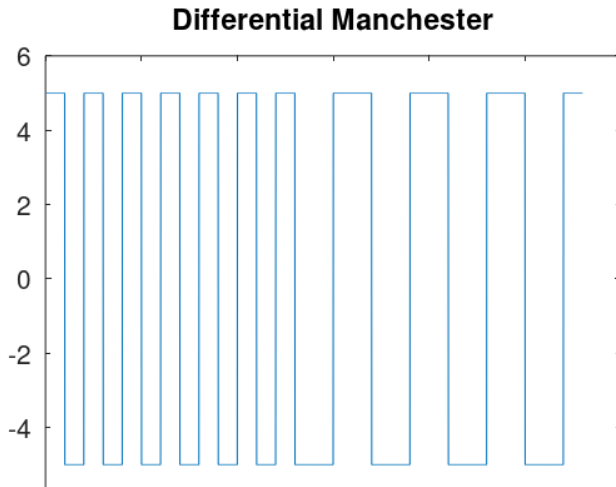
45 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



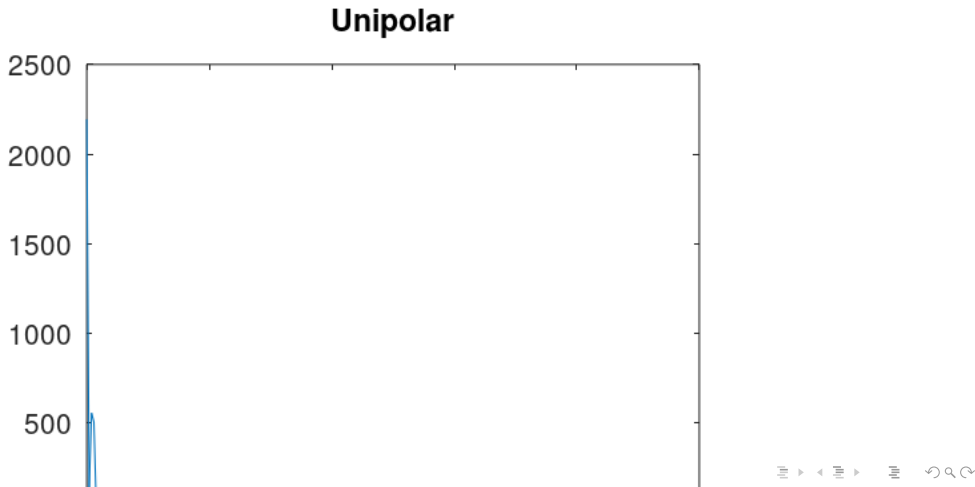
46 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



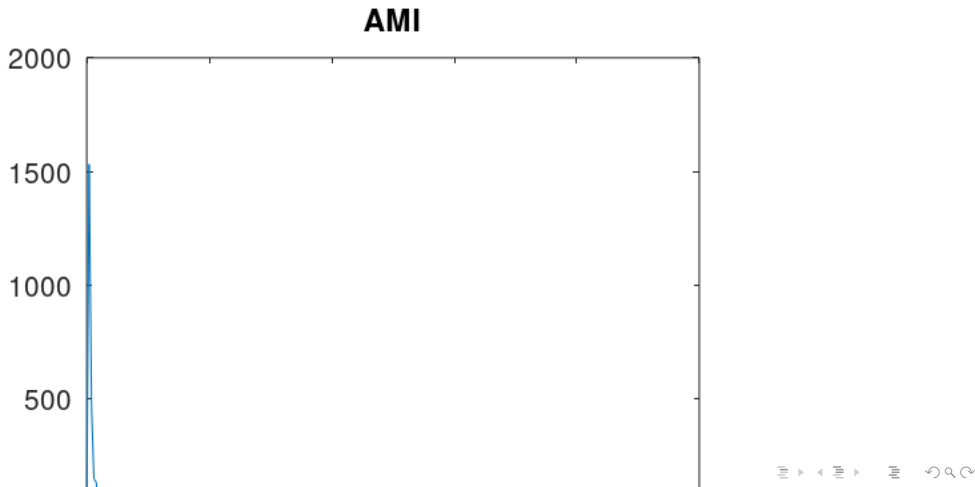
47 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



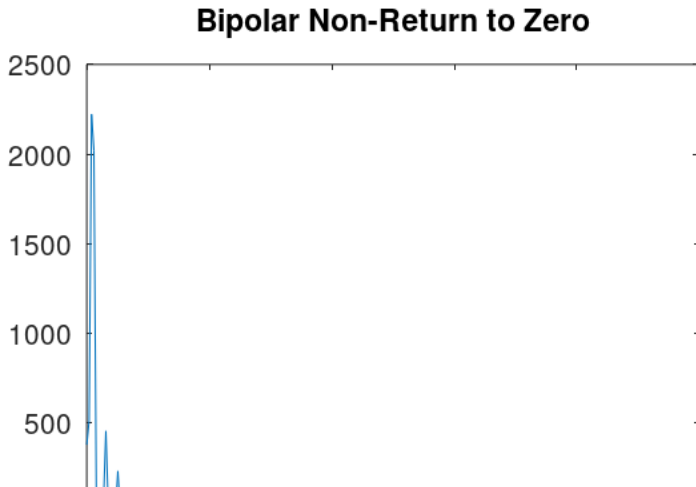
48 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



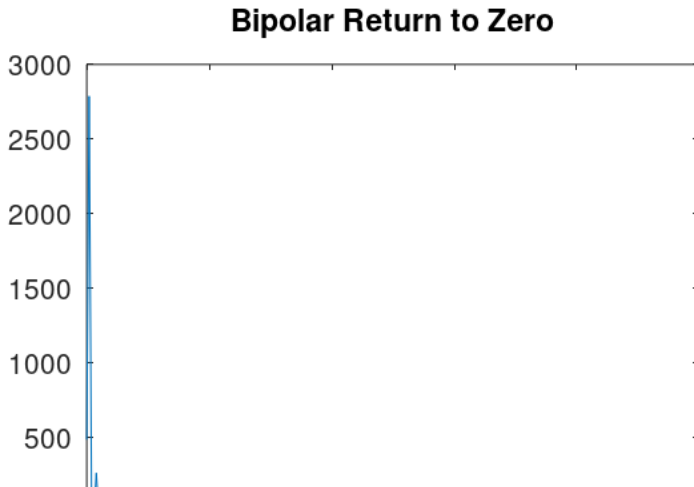
49 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



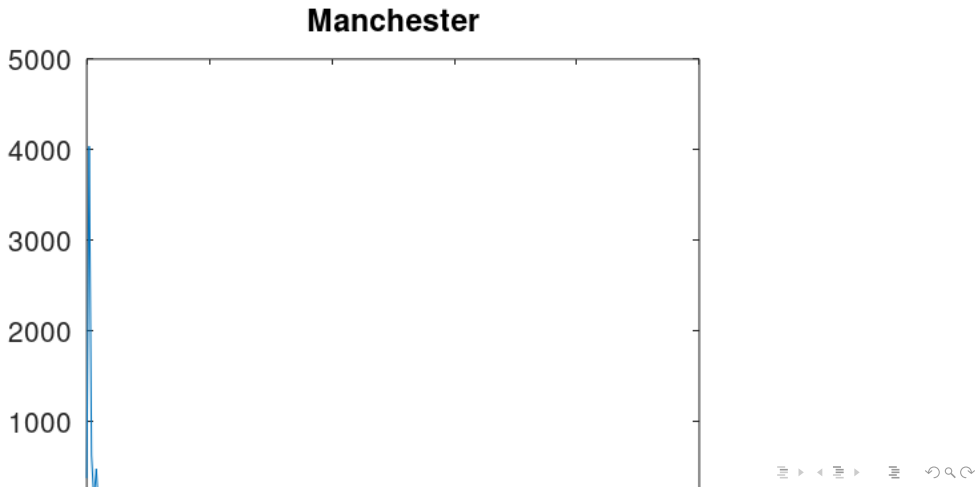
50 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



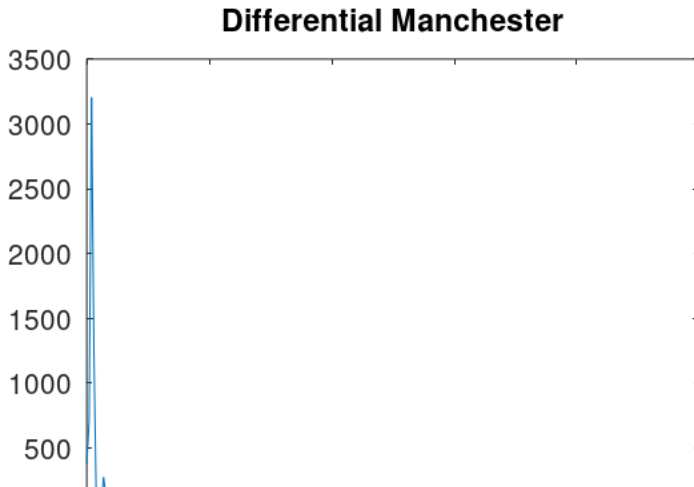
51 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



52 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



53 Кодирование сигнала. Исследование свойства самосинхронизации сигнала



54 Выводы

В ходе данной лабораторной работы я изучил методы кодирования и модуляции сигналов с помощью высокоуровневого 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.