Лабораторная работа № 1. Методы кодирования и модуляция сигналов

1.1. Цели 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.

1.2. Предварительные сведения

1.2.1. Сигнал, спектр. Преобразования Фурье

Сигнал — физическая величина, содержащая в себе определённую информацию. Примеры сигналов: звуковая волна или просто звук, вибрация, разного рода колебания, сигнал изображения и т.п.

Некоторые сигналы можно выразить в виде тригонометрических функций, зависящих от времени. Примером такой функции является синусоида:

$$f(t) = A\sin(\omega t + \vartheta),$$

где A — амплитуда, ω — угловая частота, ϑ — начальная фаза. Причём данная функция является периодически повторяющей свои значения через время T, $2T$ и т.д.

Сигнал, выражающий непрерывно изменяющуюся величину, называется *аналоговым*. Ступенчатое представление сигнала называется *дискретизацией* и может проводится как по времени, так и по значению величины сигнала. Дискретизацию по времени часто называют операцией получения *выборки*, дискретизацию по значению величины сигнала — *квантованием*.

Преобразование аналогового сигнала в цифровой посредством дискретизации по времени и по значению величины сигнала называется *аналого-цифровым преобразованием*. Полученный таким образом сигнал преобразуется в числовые значения в двоичной системе счисления, т.е. представим в виде набора нулей и единиц. Преобразование цифрового сигнала в аналоговый называется *цифро-аналоговым преобразованием*.

При аналого-цифровом преобразовании чем шире интервал дискретизации выборки и грубее квантование, тем меньше требуется данных для представления сигнала. Но если сигнал представлен слишком малым объёмом данных, то возможно потеря полезной информации, которую несёт сигнал.

Например для синусоиды теоретически обосновано, что по ряду значений выборки можно достоверно воспроизвести только одну синусоиду при условии,

что её период больше чем в два раза превышает интервал выборки. Если установить связь выборки с частотой, то по отношению к синусоиде с её наивысшей частотой f_c необходимо, чтобы частота выборки была больше $2f_c$.

Теорема Котельникова (теорема Найквиста–Шеннона или теорема отсчётов): непрерывная функция с ограниченным спектром полностью определяется своими значениями, отсчитанными через интервалы времени $\Delta t = (1/2)F$, где F — ширина спектра функции.

Иначе говоря, 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: $f > 2f_c$.

Сигнал произвольной формы можно разложить на синусоидальные составляющие с различными частотами, кратными целому числу. Совокупность этих составляющих называется *спектром*. Сумма этих составляющих формирует значение функции во временной области.

В общем виде функцию $f(t)$ можно представить в виде следующего разложения (действительного разложения функции в ряд Фурье):

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos(\omega_0 kt) + b_k \sin(\omega_0 kt)\right),$$

где ω_0 — основная угловая частота, a_k, b_k — действительные коэффициенты Фурье функции $f(t)$, определяемые следующими выражениями:

$$a_k = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos(\omega_0 kt) dt, \quad k = 0, 1, 2 \ldots,$$

$$b_k = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(\omega_0 kt) dt, \quad k = 1, 2, \ldots.$$

Колебание самого большого периода, представленное суммой косинусов и синусов, называют колебанием основной частоты или *первой гармоникой*. Колебание с периодом, равным половине основного периода, называют *второй гармоникой*, колебание с периодом, равным $1/3$ основного периода — третьей гармоникой, и т.д.

Коэффициент a_0 является постоянной величиной, выражающей среднее значение функции $f(t)$. Остальные коэффициенты Фурье выражают переменные составляющие сигнала.

Функция, симметричная относительно точки отсчёта, называется *нечётной* ($f(t) = -f(-t)$), а симметричная относительно оси ординат — *чётной* ($f(t) = f(-t)$).

Ряд Фурье чётной функции содержит только косинусы, а ряд Фурье нечётной функции содержит только синусы. Если функция $f(t)$ — нечётная, то $a_k = 0$, если функция $f(t)$ — чётная, то $b_k = 0$.

Разложение периодической функции $f(t)$ с основной угловой частотой $\omega_0 = 2\pi/T$ в *комплексный ряд Фурье* имеет вид:

$$f(t) = \sum_{-\infty}^{\infty} C_k e^{i\omega_0 kt},$$

$$C_k = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-i\omega_0 kt} dt, \quad k = 0, 1, 2, \dots .$$

Если функция $f(t)$ — нечётная, то действительная часть C_k равна 0, если функция $f(t)$ — чётная, то мнимая часть C_k равна 0.

Множество $|C_k|$ называют *спектром амплитуд* (показывает долю составляющих частот в сигнале), множество $\angle C_k$ — *спектром фаз*, множество $|C_k|^2$ — *спектром мощности*.

Импульс — одиночный сигнал конечной длительности, ограниченный неактивным (нулевым, исходным) состоянием. Сигнал состоящий из импульсов называется *импульсным сигналом*. Импульсный сигнал является периодической функцией. Возможные формы импульсов: прямоугольная, треугольная, трапециевидная, синусоидальная, экспоненциальная. В цифровой технике основным типом сигналов является периодическая последовательность прямоугольных импульсов.

Для импульсных сигналов отношение периода повторения импульсов T к длительности одиночного импульса t называется *скважностью импульсов*. При увеличении скважности (т.е. при уменьшении длительности импульса при фиксированном периоде повторения) значение огибающей спектра на нулевой частоте уменьшается.

Дискретное преобразование Фурье сигнала $\{f_0, f_1, \dots, f_{N-1}\}$ определяется соотношением:

$$C_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-i(2\pi/N)kj}, \quad k = 0, 1, 2, \dots, N-1.$$

Обратное дискретное преобразование Фурье выражает каждый компонент вектора $\vec{f}$:

$$f_j = \sum_{k=0}^{N-1} C_k e^{i(2\pi/N)kj}, \quad j = 0, 1, 2, \dots, N-1,$$

или при $C_k = A_k + iB_k$:

$$f_j = \sum_{k=0}^{N-1} \left(A_k \cos\left(\frac{2\pi}{N} kj\right) - B_k \sin\left(\frac{2\pi}{N} kj\right) \right), \quad j = 0, 1, 2, \ldots, N-1.$$

Под *быстрым преобразованием Фурье* понимают алгоритм эффективного вычисления с использованием закономерностей, скрытых внутри матрицы, выражающей дискретное преобразование Фурье.

1.2.2. Модуляция

Модуляция — процесс изменения одного или нескольких параметров высокочастотного несущего колебания по закону низкочастотного информационного сигнала (сообщения).

Несущая — высокочастотное колебание, выполняющее роль переносчика информации, заложенной в управляющем (модулирующем) сигнале.

В качестве несущего колебания наиболее часто используют *гармоническое колебание*. В зависимости от того, какой из параметров несущего колебания — амплитуда, частота или начальная фаза несущего колебания изменяется по закону передаваемого сообщения, различают виды модуляции, соответственно, *амплитудная*, *частотная* или *фазовая*.

Сигнал, получаемый в процессе модуляции, называют *модулированным колебанием*, или радиосигналом.

В процессе *амплитудной модуляции* амплитуда U_0 несущего колебания $u_0(t) = U_0 \cos(\omega t + \varphi)$ перестает быть постоянной и изменяется по закону передаваемого сообщения. Амплитуда $U(t)$ несущего колебания может быть связана с передаваемым сообщением соотношением

$$U(t) = U_0 + k_A s(t),$$

где U_0 — амплитуда несущего колебания в отсутствии сообщения (немодулированное колебание); $s(t)$ — функция, зависящая от времени, соответствующая передаваемому сообщению (ее называют модулирующим сигналом); k_A — коэффициент пропорциональности, отражающий степень влияния модулирующего сигнала на величину изменения амплитуды результирующего сигнала (модулированного колебания).

Выражение для амплитудно-модулированного сигнала в общем случае имеет вид

$$u(t) = [U_0 + k_A s(t)] \cos(\omega_0 t + \varphi).$$

При *фазовой модуляции* значение фазового угла постоянной несущей частоты колебаний ω_0 пропорционально амплитуде модулирующего сигнала $s(t)$. Соответственно, уравнение ФМ–сигнала определяется выражением

$$u(t) = U \cos[\omega_0 t + ks(t)],$$

где k — коэффициент пропорциональности.

Частотная модуляция характеризуется линейной связью модулирующего сигнала с мгновенной частотой колебаний, при которой мгновенная частота колебаний образуется сложением частоты высокочастотного несущего колебания ω_0 со значением амплитуды модулирующего сигнала с определенным коэффициентом пропорциональности:

$$\omega(t) = \omega_0 + ks(t).$$

1.2.3. Кодирование сигнала

Одной из основных задач физического уровня модели OSI является преобразование данных в электромагнитные сигналы, и наоборот. Переход от электромагнитых импульсов к последовательности бит называют *кодированием сигнала*.

Способ представления исходного кода определёнными сигналами определяется форматом кода:

- *NRZ (Non Return to Zero)* — простейший двухуровневый код: логической единице соответствует верхний уровень, логическому нулю — нижний, переходы электрического сигнала происходят на границе битов;
- AMI-код (используется в телефонии): биты 0 представляются нулевым напряжением (0 В); биты 1 представляются поочерёдно значениями -U или +U (В);
- *RZ (Return to Zero)* — трёхуровневый код: обеспечивает возвращение к нулю после передачи каждого бита информации, в центре которого всегда есть переход, причём логической единице соответствует отрицательный импульс, логическому нулю — положительный;
- *манчестерский код* — двухуровневый код: логической единице соответствует переход вниз в центре бита, логическому нулю — переход вверх;
- *дифференциальный манчестерский код* — двухуровневый код: в течение битового интервала (времени передачи одного бита) уровень сигнала может меняться дважды, причём обязательно происходит изменение уровня в середине интервала и получается, что при передаче нуля в начале битового интервала происходит перепад уровней, а при передаче единицы такой перепад отсутствует.

1.2.4. Высокоуровневый интерпретируемый язык программирования Octave

Octave — высокоуровневый интерпретируемый язык программирования, предназначенный для решения задач вычислительной математики.

Интерпретатор Octave запускается из терминала операционной системы с помощью команды `octave` для работы с помощью консоли или `octave --gui` для работы с оконным интерфейсом.

Octave имеет два режима работы: *терминальный* и *программный*. В терминальном режиме отдельные команды последовательно вводятся в окне интерпретатора (или в командном окне оконного интерфейса Octave). В программном

режиме создается текстовый файл (с расширением **.m**), в котором хранятся последовательно выполняемые команды, впоследствии запускаемые на выполнение в среде Octave.

В окне интерпретатора Octave пользователь может вводить как отдельные команды языка Octave, так и группы команд, объединяемые в программы. Если строка заканчивается символом «;», то результаты на экран не выводятся. Если в конце строки символ «;» отсутствует, то результаты работы выводятся на экран. Текст в стоке, который идёт после символа %, является строкой комментария и интерпретатором не обрабатывается.

Пример простых операций в окне интерпретатора Octave:

```
>> % простейшие арифметические  операции над числами:
>> 2+3*2
ans = 8
>> % пример с подавлением вывода результата на экран:
>> a=3+2;
>> b=a+1
b = 6
```

Здесь >> — знак приглашения Octave для ввода команд в окне интерпретатора команд.

Простейшие арифметические операции в Octave:

`+` — сложение;
`-` — вычитание;
`*` — умножение;
`/` — деление слева направо;
`\` — деление справа налево;
`^` — возведение в степень.

Для определения переменной необходимо набрать *имя переменной*, символ «=» и *значение переменной*, где знак равенства – это оператор присваивания:

```
имя_переменной = значение_выражения
```

Система различает большие и малые буквы в именах переменных. Выражение в правой части оператора присваивания может быть числом, арифметическим выражением, строкой символов или символьным выражением. Если речь идет о символьной или строковой переменной, то выражение в правой части оператора присваивания следует брать в одинарные кавычки.

Если команда не содержит знака присваивания, то по умолчанию вычисленное значение присваивается специальной системной переменной `ans`. Причем полученное значение можно использовать в последующих вычислениях, но важно помнить, что значение `ans` изменяется после каждого вызова команды без оператора присваивания.

Системные переменные:

- `ans` — результат последней операции без знака присваивания;
- `i` — мнимая единица ($\sqrt{-1}$);
- `pi` — число π (3.141592653589793);
- `e` — число e (экспонента 2.71828183);
- `inf` — машинный символ бесконечности (∞);
- `NaN` — неопределенный результат;

- `realmin` — наименьшее число с плавающей точкой (2.2251e-308);
- `realmax` — наибольшее число с плавающей точкой (1.7977e+308).

Все перечисленные переменные можно использовать в математических выражениях.

Команда `clear` предназначена для уничтожения определения одной или нескольких переменных:

```
clear имя_переменной
```

Операции отношения выполняют сравнение двух операндов и определяют, истинно выражение или ложно:

`<` — меньше;
`>` — больше;
`= =` — равно;
`~=` —не равно;
`<=` — меньше или равно;
`>=` — больше или равно.

В общем виде обращение к функции в Octave имеет вид

```
имя_переменной = имя_функции(аргумент)
```

или

```
имя_функции(аргумент)
```

Если имя переменной указано, то ей будет присвоен результат работы функции. Если же оно отсутствует, то значение вычисленного функцией результата присваивается системной переменной `ans`.

Пример работы с тригонометрическими функциями:

```
>> % Определение значения аргумента:
>> x=pi/2;
>> % Вызов функции:
>> y=sin(x)
y = 1
>> % Вызов функции:
>> cos(pi/3)
ans = 0.50000
```

Перечень некоторых встроенных функций Octave (с полным перечнем можно ознакомиться в документации):

- `sin(x)` — синус числа x;
- `cos(x)` — косинус числа x;
- `tan(x)` — тангенс числа x;
- `cot(x)` — котангенс числа x;
- `sec(x)` — секанс числа x;
- `csc(x)` — косеканс числа x;
- `exp(x)` — экспонента числа x;
- `log(x)` — натуральный логарифм числа x;
- `round(x)` — обычное округление числа x до ближайшего целого;
- `rem(x, y)` — вычисление остатка от деления x на y;
- `sign(x)` — сигнум-функция числа x, выдаёт 0, если $= 0$, -1 — при $x < 0$ и 1 при $x > 0$;
- `sqrt(x)` — корень квадратный из числа x;
- `abs(x)` — модуль числа x;

- `log10(x)` — десятичный логарифм от числа x;
- `log2(x)` — логарифм по основанию два от числа x;
- `real(Z)` — выдаёт действительную часть комплексного аргумента Z;
- `imag(Z)` — выдаёт мнимую часть комплексного аргумента Z;
- `angle(Z)` — вычисляет значение аргумента комплексного числа Z в радианах от $-\pi$ до π;
- `conj(Z)` — выдаёт число, комплексно сопряжённое Z.

Синтаксис функции, определяемой пользователем:

```
function name1 [, name2, ...] = fun(var1 [, var2, ...])
```

Здесь `name1 [, name2, ...]` — список выходных параметров, то есть переменных, которым будет присвоен конечный результат вычислений, `fun` — имя функции, `var1 [, var2, ...]` — входные параметры.

Все имена переменных внутри функции, а также имена из списка входных и выходных параметров воспринимаются системой как локальные, т.е. эти переменные считаются определенными только внутри функции.

Программы и функции в Octave могут быть созданы при помощи текстового редактора и сохранены в виде файла с расширением `.m` или `.M`. Но при создании и сохранении функции следует помнить, что её имя должно совпадать с именем файла. Программу можно запустить на выполнение, указав имя файла, в котором она сохранена. Обращение к функции осуществляется так же, как и к любой другой встроенной функции системы, то есть с указанием входных и выходных параметров. Вызвать функцию можно из командной строки или использовать ее как один из операторов программы.

Массив — множественный тип данных, состоящий из фиксированного числа элементов одного типа. Как и любой другой переменной, массиву должно быть присвоено имя.

Самый простой способ задать одномерный массив в Octave имеет вид

```
имя_массива = Xn:dX:Xk
```

Здесь `Xn` — значение первого элемента массива, `Xk` — значение последнего элемента массива, `dX` — шаг, с помощью которого формируется каждый следующий элемент массива, т.е. значение второго элемента составит `Xn+dX`, третьего `Xn+dX+dX` и так далее до `Xk`.

Примеры создания массивов:

```
>> % Массив целых чисел от 1 до 5:
>> A=1:4
A =
   1 2 3 4
>> % Массив целых чисел от 2 до 10 с шагом 2:
>> B=2:2:10
B =
   2 4 6 8 10
>> % Массив от -3.5 до 0.5 с шагом 0,5:
>> xn=-3.5;xk=3.5;dx=0.5;
>> X=xn:dx:xk
X =
   Columns 1 through 8:
   -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0
```

```
    Columns 9 through 15:
    0.5 1.0 1.5 2.0 2.5 3.0 3.5
>> % Операция транспонирования:
>> A'
ans =
 1
 2
 3
 4
```

Обратиться к элементу массива можно, указав имя массива и порядковый номер элемента в круглых скобках:

```
>> % Массивы x и y:
>> x=[2 4 6 8 10];
>> y=[-1.2 3.4 -0.8 9.1 5.6 -7.3];
>> % Значение первого элемента массива x:
>> x(1)
ans = 2
>> % Значение пятого элемента массива y:
>> y(5)
ans = 5.6000
>> % Пример операций с элементами массива:
>> x(1)/2+y(3)^2-x(4)/y(5)
ans = 0.21143
```

Ввод элементов двумерного массива (матрицы) также осуществляется в квадратных скобках, при этом элементы строки отделяются друг от друга пробелом или запятой, а строки разделяются между собой точкой с запятой. Обратиться к элементу матрицы можно, указав после имени матрицы, в круглых скобках, через запятую, номер строки и номер столбца, на пересечении которых элемент расположен:

```
>> % Матрица M:
>> M=[2 4 6;1 3 5;7 8 9]
M =
    2 4 6
    1 3 5
    7 8 9
>> % Обращение к элементу матрицы из 1-й строки и 2-го
 ↪  столбца:
>> M(1,2)
ans = 4
>> % Обращение к элементу матрицы из 3-й строки и 1-го
 ↪  столбца:
>> M(3,1)
ans = 7
```

Массивы применяются в том числе и для построения графиков посредством функции `plot()`. Для того чтобы построить двумерный график функции $f(x)$, необходимо сформировать два массива x и y одинаковой размерности, а затем обратиться к функции `plot()`.

Синтаксис функции `plot()`:

```
plot(x1, y1, s1, x2, y2, s2, ..., xn, yn, sn)
```

Здесь `x1, x2, ..., xn` — массивы абсцисс графиков; `y1, y2, ..., yn` — массивы ординат графиков; `s1, s2, ..., sn` — строка форматов, определяющая параметры линии и, при необходимости, позволяющая вывести легенду.

В строке форматов могут участвовать символы, отвечающие за тип линии, маркер и его размер, цвет линии и вывод легенды. За сплошную линию отвечает символ «-». Цвет линии определяется буквой латинского алфавита: `y` — жёлтый, `m` — розовый, `c` — голубой, `r` — красный, `g` — зелёный, `b` — синий, `w` — белый. Некоторые символы маркера: `.` — точка, * — звёздочка, `x` — крестик, + — плюс, `o` — незакрашенный круг, `p` — незакрашенный квадрат.

Если повторно обратиться к функции `plot()`, то в этом же окне будет стёрт первый график и нарисован второй. Для построения нескольких графиков в одной системе координат можно поступить одним из следующих способов:

1) обратиться к функции `plot()` следующим образом:

   ```
   plot(x1,y1,x2,y2,... xn,yn),
   ```

   где `x1`, `y1` — массивы абсцисс и ординат первого графика, `x2`, `y2` — массивы абсцисс и ординат второго графика, ..., `xn`, `yn` — массивы абсцисс и ординат n-го графика;
2) каждый график изображать с помощью функции `plot(x,y)`, но перед обращением к функциям `plot(x2,y2)`, `plot(x3,y3)`, ..., `plot(xn,yn)` вызвать команду `hold on`, которая блокирует режим очистки окна.

С помощью встроенного в Octave менеджера пакетов (pkg) есть возможность расширить фаункционал Octave путём установки дополнительных пакетов. С перечнем пакетов расширений можно ознакомиться на ресурсах:

- `https://gnu-octave.github.io/packages/`
- `https://octave.sourceforge.io/packages.php`

Устанавливать пакеты можно локально в каталоге пользователя или глобально относительно операционной системы, из центрального репозитория или с стороннего ресурса.

Для указания системе места установки пакета, например, локально в каталоге пользователя, в окне инерпретатора команд следует ввести команду:

```
pkg prefix -local
```

Для просмотра списка имеющихся в центральном репозитории пакетов используется команда

```
pkg list -forge
```

Для просмотра списка локально установленных пакетов используется команда

```
pkg list
```

Для установки пакета из центрального репозитория используется команда:

```
pkg install <package_name> -forge
```

После установки вашего пакета для его использования в разрабатываемом коде следует его загрузить:

```
pkg load <package_name>
```

Если требуется удалить пакет, то используется команда

```
pkg uninstall <package_name>
```

1.3. Задания для выполнения

1.3.1. Построение графиков в Octave

1.3.1.1. Постановка задачи

1. 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`.
2. Добавить график функции $y = \cos x + \frac{1}{3}\cos 3x + \frac{1}{5}\cos 5x$ на интервале $[-10; 10]$. График экспортировать в файлы формата `.eps`, `.png`.

1.3.1.2. Порядок выполнения работы

1. Запустите в вашей ОС Octave с оконным интерфейсом.
2. Перейдите в окно редактора. Воспользовавшись меню или комбинацией клавиш ctrl+n создайте новый сценарий. Сохраните его в ваш рабочий каталог с именем, например, `plot_sin.m`.
3. В окне редактора повторите следующий листинг по построению графика функции $y = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x$ на интервале $[-10; 10]$:

```
% 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");
```

4. Запустите сценарий на выполнение (воспользуйтесь соответствующим меню окна редактора или клавишей F5). В качестве результата выполнения кода должно открыться окно с построенным графиком (рис. 1.1) и в вашем рабочем каталоге должны появиться файлы с графиками в форматах `.eps`, `.png`.

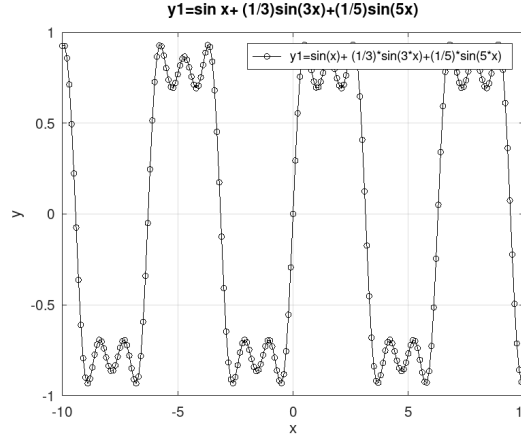

Рис. 1.1. График функции $y = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x$ **на интервале** $[-10; 10]$

5. Сохраните сценарий под другим названием и измените его так, чтобы на одном графике располагались отличающиеся по типу линий графики функций $y1 = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x, y2 = \cos x + \frac{1}{3}\cos 3x + \frac{1}{5}\cos 5x$, например как изображено на рис. 1.2.

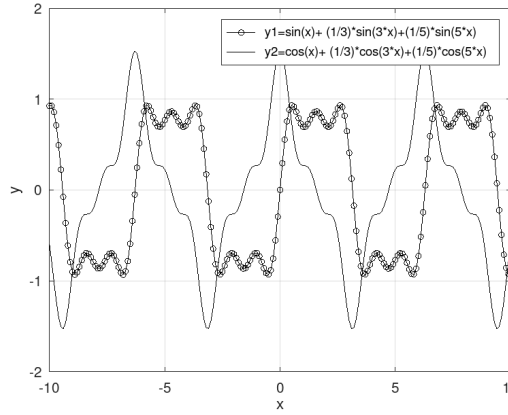

Рис. 1.2. График функций $y1$ **и** $y2$ **на интервале** $[-10; 10]$

1.3.2. Разложение импульсного сигнала в частичный ряд Фурье

1.3.2.1. Постановка задачи

1. Разработать код m-файла, результатом выполнения которого являются графики меандра (рис. 1.3), реализованные с различным количеством гармоник.

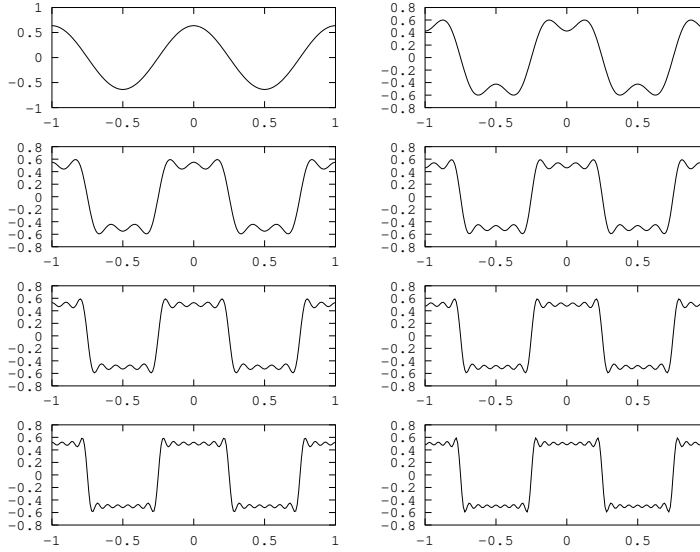

Рис. 1.3. Графики меандра, содержащего различное число гармоник

1.3.2.2. Порядок выполнения работы

1. Создайте новый сценарий и сохраните его в ваш рабочий каталог с именем, например, `meandr.m`.
2. В коде созданного сценария задайте начальные значения:

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

```
A=1;
% период:
T=1;
```

3. Разложение импульсного сигнала в форме меандра в частичный ряд Фурье можно задать формулой

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

или формулой

$$s(t) = \frac{A}{2} + \frac{2A}{\pi}\left(\sin\left(\frac{2\pi}{T}t\right) + \frac{1}{3}\sin\left(3\frac{2\pi}{T}t\right) + \frac{1}{5}\sin\left(5\frac{2\pi}{T}t\right) + ...\right).$$

т.е. в спектре присутствуют только нечётные гармоники.
Гармоники, образующие меандр, имеют амплитуду, обратно пропорциональную номеру соответствующей гармоники в спектре:

```
% 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);
```

Далее задаём массив значений гармоник массив элементов ряда:

```
% массив гармоник:
harmonics=cos(2 * pi * nh' * t/T);
% массив элементов ряда:
s1=harmonics.*repmat(Am',1,length(t));
```

4. 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:

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

5. Экспортируйте полученный график в файл в формате `.png`.
6. Скорректируйте код для реализации меандра через синусы. Получите соответствующие графики.

1.3.3. Определение спектра и параметров сигнала

1.3.3.1. Постановка задачи

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?

1.3.3.2. Порядок выполнения работы

1. В вашем рабочем каталоге создайте каталог `spectre1` и в нём новый сценарий с именем, `spectre.m`.
2. В коде созданного сценария задайте начальные значения:

```
% 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;
```

3. Далее в коде задайте два синусоидальных сигнала разной частоты:

```
% Два сигнала разной частоты:
signal1 = a1*sin(2*pi*t*f1);
signal2 = a2*sin(2*pi*t*f2);
```

4. Постройте графики сигналов (рис. 1.4):

```
% 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';
```

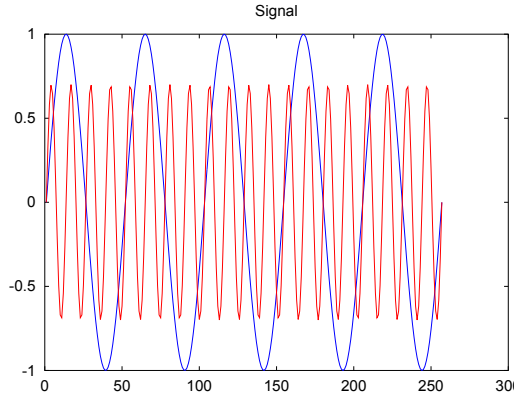

Рис. 1.4. Два синусоидальных сигнала разной частоты

5. С помощью быстрого преобразования Фурье найдите спектры сигналов (рис. 1.5), добавив в файл `spectre.m` следующий код:

```
% 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';
```

6. Учитывая реализацию преобразования Фурье, скорректируйте график спектра (рис. 1.6): отбросьте дублирующие отрицательные частоты, а также примите в расчёт то, что на каждом шаге вычисления быстрого преобразования Фурье происходит суммирование амплитуд сигналов. Для этого добавьте в файл `spectre.m` следующий код:

```
% 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';
```

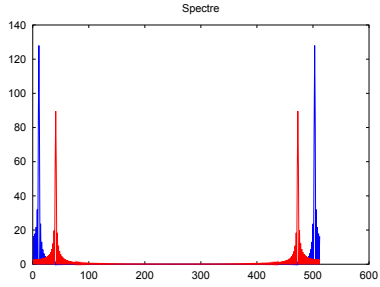

Рис. 1.5. График спектров синусоидальных сигналов

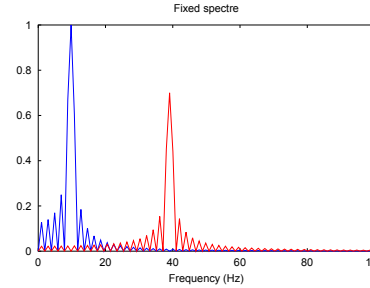

Рис. 1.6. Исправленный график спектров синусоидальных сигналов

7. Найдите спектр суммы рассмотренных сигналов (рис. 1.7), 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 должен получится аналогичный предыдущему результат (рис. 1.8), т.е. спектр суммы сигналов должен быть равен сумме спектров сигналов, что вытекает из свойств преобразования Фурье.

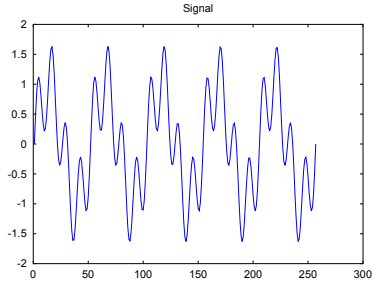

Рис. 1.7. Суммарный сигнал

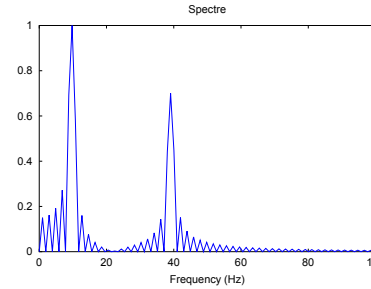

Рис. 1.8. Спектр суммарного сигнала

1.3.4. Амплитудная модуляция

1.3.4.1. Постановка задачи

Продемонстрировать принципы модуляции сигнала на примере аналоговой амплитудной модуляции (рис.1.9).

1.3.4.2. Порядок выполнения работы

1. В вашем рабочем каталоге создайте каталог `modulation` и в нём новый сценарий с именем `am.m`.
2. Добавьте в файле `am.m` 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 = 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 off
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';
```

В результате получаем, что спектр произведения представляет собой свёртку спектров (рис. 1.10).

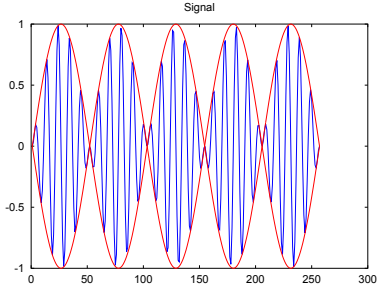

Рис. 1.9. Сигнал и огибающая при амплитудной модуляции

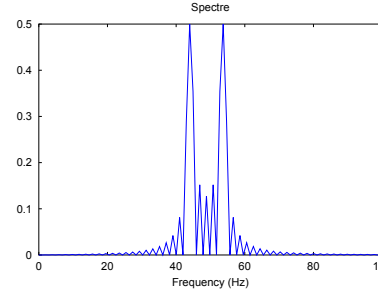

Рис. 1.10. Спектр сигнала при амплитудной модуляции

1.3.5. Кодирование сигнала. Исследование свойства самосинхронизации сигнала

1.3.5.1. Постановка задачи

По заданных битовых последовательностей требуется получить кодированные сигналы для нескольких кодов, проверить свойства самосинхронизуемости кодов, получить спектры.

1.3.5.2. Порядок выполнения работы

1. В вашем рабочем каталоге создайте каталог `coding` и в нём файлы `main.m`, `maptowave.m`, `unipolar.m`, `ami.m`, `bipolarnrz.m`, `bipolarrz.m`, `manchester.m`, `diffmanc.m`, `calcspectre.m`.
2. В окне интерпретатора команд проверьте, установлен ли у вас пакет расширений `signal`:
   ```
   >> pkg list
   ```
   Если пакет не установлен, то установите его:
   ```
   >> pkg list -forge
   >> pkg install control signal
   ```
3. В файле `main.m` подключите пакет `signal` и задайте входные кодовые последовательности:
   ```
   % coding/main.m
   % Подключение пакета signal:
   pkg load signal;
   ```

```
% Входная кодовая последовательность:
data=[0 1 0 0 1 1 0 0 0 1 1 0];
% Входная кодовая последовательность для проверки
 ↪  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");
```

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

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

% Кодирование ami
wave=ami(data);
plot(wave)
title('AMI');
print 'signal/ami.png';

% Кодирование NRZ
wave=bipolarnrz(data);
plot(wave);
title('Bipolar Non-Return to Zero');
print 'signal/bipolarnrz.png';

% Кодирование RZ
wave=bipolarrz(data);
plot(wave)
title('Bipolar Return to Zero');
print 'signal/bipolarrz.png';

% Манчестерское кодирование
wave=manchester(data);
plot(wave)
title('Manchester');
print 'signal/manchester.png';
```

```
% Дифференциальное манчестерское кодирование
wave=diffmanc(data);
plot(wave)
title('Differential Manchester');
print 'signal/diffmanc.png';
```

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

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

% Кодирование AMI
wave=ami(data_sync);
plot(wave)
title('AMI');
print 'sync/ami.png';

% Кодирование NRZ
wave=bipolarnrz(data_sync);
plot(wave);
title('Bipolar Non-Return to Zero');
print 'sync/bipolarnrz.png';

% Кодирование RZ
wave=bipolarrz(data_sync);
plot(wave)
title('Bipolar Return to Zero');
print 'sync/bipolarrz.png';

% Манчестерское кодирование
wave=manchester(data_sync);
plot(wave)
title('Manchester');
print 'sync/manchester.png';

% Дифференциальное манчестерское кодирование
wave=diffmanc(data_sync);
plot(wave)
title('Differential Manchester');
print 'sync/diffmanc.png';
```

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

```
% 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';
```

4. В файле `maptowave.m` пропишите 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);
```

5. В файлах `unipolar.m`, `ami.m`, `bipolarnrz.m`, `bipolarrz.m`, `manchester.m`, `diffmanc.m` пропишите 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);
```

6. В файле `calcspectre.m` пропишите 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)');
```

7. Запустите главный скрипт `main.m`. В каталоге `signal` должны быть получены файлы с графиками кодированного сигнала (рис. 1.11–1.16), в каталоге `sync` — файлы с графиками, иллюстрирующими свойства самосинхронизации (рис. 1.17–1.22), в каталоге `spectre` — файлы с графиками спектров сигналов (рис. 1.23–1.28).

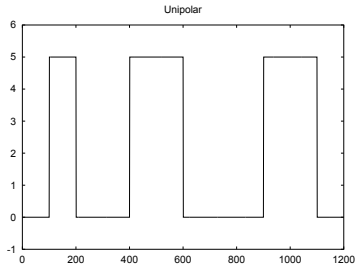

Рис. 1.11. Униполярное кодирование

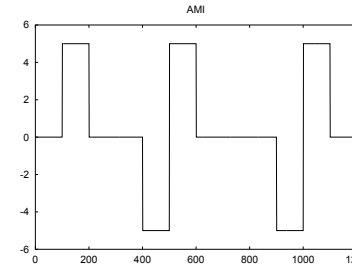

Рис. 1.12. Кодирование AMI

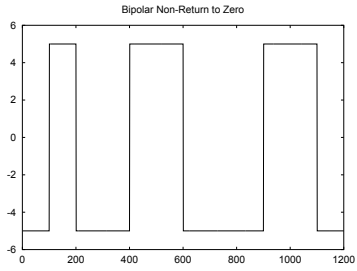

Рис. 1.13. Кодирование NRZ

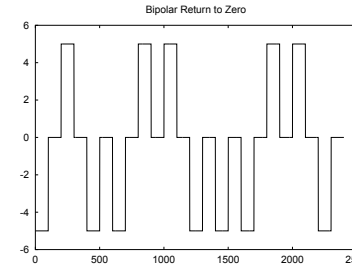

Рис. 1.14. Кодирование RZ

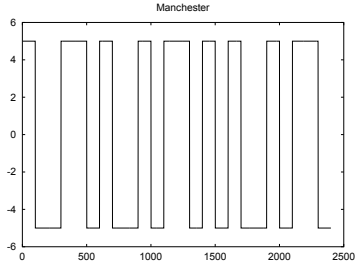

Рис. 1.15. Манчестерское кодирование

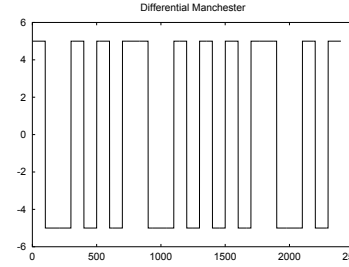

Рис. 1.16. Дифференциальное манчестерское кодирование

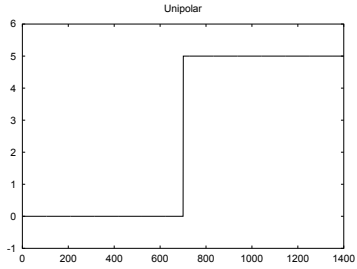

Рис. 1.17. Униполярное кодирование: нет самосинхронизации

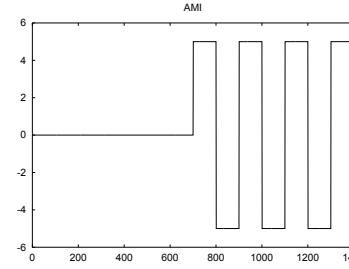

Рис. 1.18. Кодирование AMI: самосинхронизация при наличии сигнала

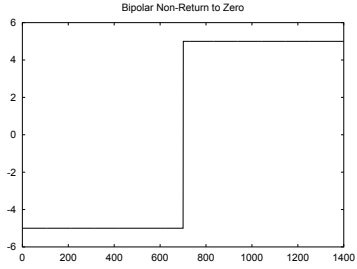

Рис. 1.19. Кодирование NRZ: нет самосинхронизации

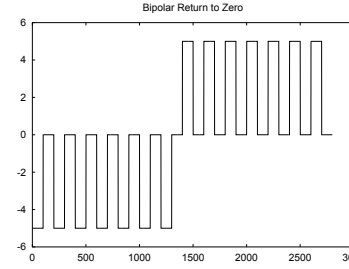

Рис. 1.20. Кодирование RZ: есть самосинхронизация

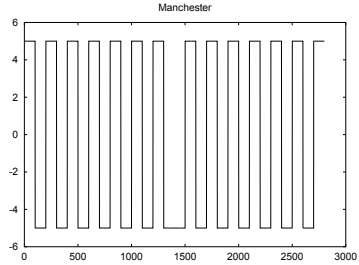

Рис. 1.21. Манчестерское кодирование: есть самосинхронизация

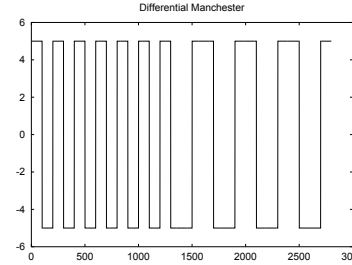

Рис. 1.22. Дифференциальное манчестерское кодирование: есть самосинхронизация

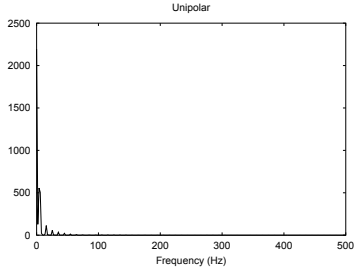

Рис. 1.23. Униполярное кодирование: спектр сигнала

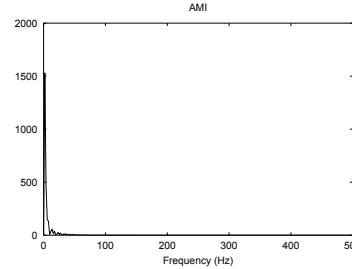

Рис. 1.24. Кодирование AMI: спектр сигнала

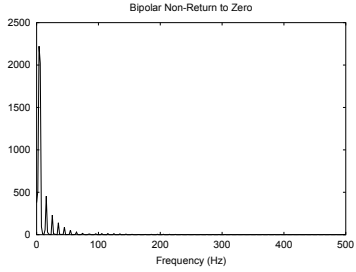

Рис. 1.25. Кодирование NRZ: спектр сигнала

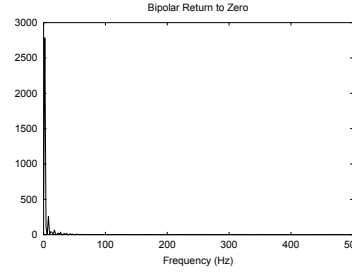

Рис. 1.26. Кодирование RZ: спектр сигнала

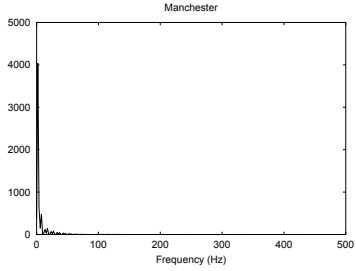

Рис. 1.27. Манчестерское кодирование: спектр сигнала

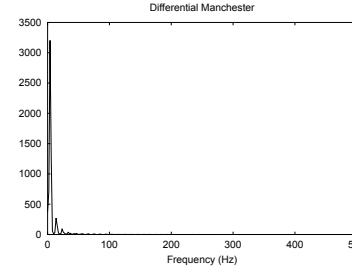

Рис. 1.28. Дифференциальное манчестерское кодирование: спектр сигнала

1.4. Содержание отчёта

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка задания работы.
3. Описание результатов выполнения задания:
   - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
   - пояснения по отображаемой информации согласно заданию
4. Выводы, согласованные с заданием работы.