Лабораторная работа №1. Дискретизация и квантование сигналов.

Третьяк Илья Дмитриевич (Tretyak0ID@gmail.com)

ПМ-31. Вариант 20 + 1==0+1=1 (mod 4)

Часть 1. Исследование эффектов дискретизации.

Цель работы:

Исследование влияния частоты дискретизации на спектр дискретных сигналов.

Ход работы:

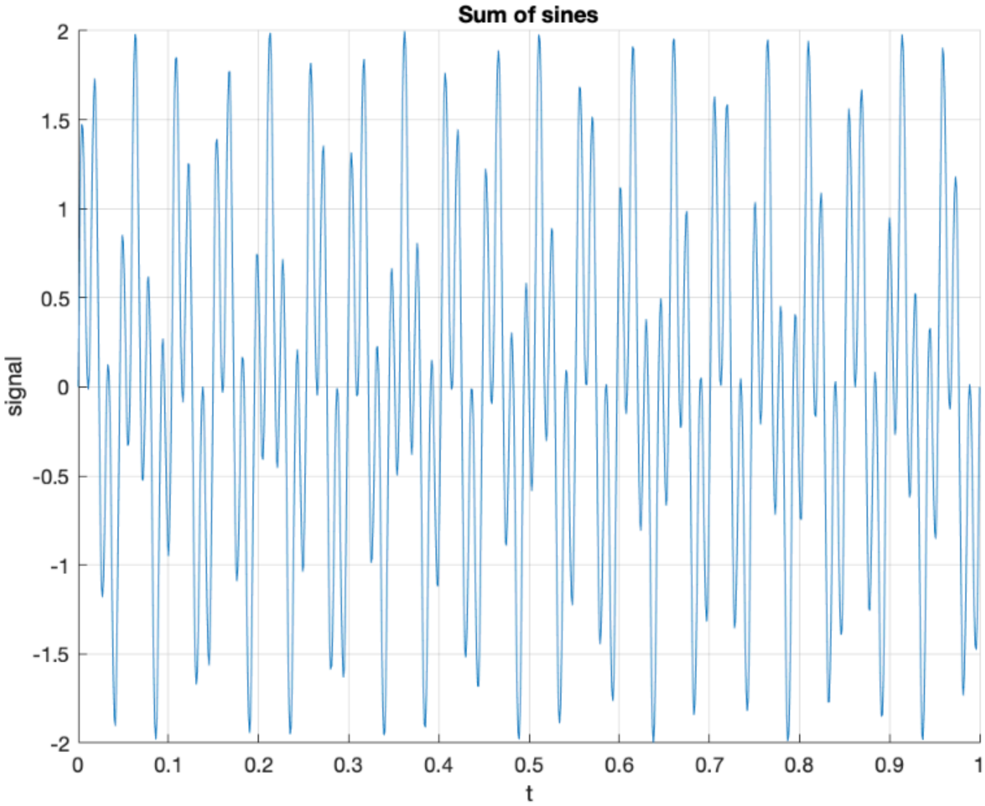
1. Синтезировать конечный сигнал $x(t) = sin(2\pi\nu_1 t) + sin(2\pi\nu_2 t),\ 0 \leq t \leq 1.$

Вариант 1 предполагает значения $\nu_1 = 20,\ \nu_2 = 67.$ Для синтеза сигнала использовалась функция MATLAB GenerateSignal

```
function output_signal = GenerateSignal(t_min,t_max,t_step,nu)
%Функция генерации сигнала представляющего собой сумму синусов с набором
%частот nu (массив частот)

    t  = [t_min:t_step:t_max];
    output_signal = sin(2*pi*nu(1).*t) + sin(2*pi*nu(2).*t);

    hold on; grid on;
    plot(t, output_signal)
    title('Sum of sines'); xlabel('t'); ylabel('signal');
end
```

Для реализации в main-файле использовалась следующая реализация с иллюстрацией синтезированного сигнала:

```
clear; clc;
%Входные-параметры-для-генерации-сигнала
t_min  = 0;
t_max  = 1;
t_step = 10^(-3);
nu = [20, 67];

%Генерация-сигнала
signal = GenerateSignal(t_min,t_max,t_step,nu);
```



2. Определить допустимые значения частоты дискретизации f_s для $x(t)$.

Для определения частоты дискретизации исподьзуем соотношение $f_s > 2F_{max}$. Тогда задача сводится к определению частоты F_{max}.

- Аналитически

$$S(\nu) = \int\limits_{-\infty}^{+\infty} [sin(2\pi\nu_1 t) + sin(2\pi\nu_2 t)]e^{-2\pi i\nu t}dt = \int\limits_{-\infty}^{+\infty} sin(2\pi\nu_1 t)e^{-2\pi i\nu t}dt + \int\limits_{-\infty}^{+\infty} sin(2\pi\nu_2 t)e^{-2\pi i\nu t}dt =$$

$$\frac{1}{2i}\int\limits_{-\infty}^{+\infty} (e^{2\pi i\nu_1 t} - e^{-2\pi i\nu_1 t})e^{-2\pi i\nu t}dt + \frac{1}{2i}\int\limits_{-\infty}^{+\infty} (e^{2\pi i\nu_2 t} - e^{-2\pi i\nu_2 t})e^{-2\pi i\nu t}dt =$$

$$\frac{1}{2i}(\int\limits_{-\infty}^{+\infty} e^{2\pi i(\nu_2-\nu)t}dt + \int\limits_{-\infty}^{+\infty} e^{2\pi i(\nu_1-\nu)t}dt - \int\limits_{-\infty}^{+\infty} e^{-2\pi i(\nu_2+\nu)t}dt - \int\limits_{-\infty}^{+\infty} e^{-2\pi i(\nu_1+\nu)t}dt) =$$

$$\frac{1}{2i}(\delta(\nu-\nu_2) + \delta(\nu-\nu_1) - \delta(\nu+\nu_2) - \delta(\nu+\nu_2))$$

Отсюда можно заключить о том, что ненулевые значения частотный спектр исппледуемого сигнала принимает лишь в точках $\nu = \pm\nu_1$ и $\nu = \pm\nu_2$. Следовательно, носитель $S(\nu)$ это отрезок $[-max\{|\nu_1|, |\nu_2|\}, max\{|\nu_1|, |\nu_2|\}]$. Окончательно получаем:

$$F_{max} = max\{|\nu_1|, |\nu_2|\} = max\{20, 67\} = 67$$

Следовательно допустимые значения частот $\{f_\partial \mid f_\partial > 2 * 67 = 134\}$.

3. Построить по отсчетам графики дискретного сигнала и его спектра при нескольких различных частотах дискретизации (больше и меньше граничной частоты дискретизации).

Для дискретизации сигнала была использована следующая функция:

```
function [dot, disc_signal] = SignalSamp(sampling_frequency, t_min, t_max, nu)
%Функция, производящая дискретизацию сигнала, при этом дискретизированный
%сигнал должен быть реализован в виде m-функции GenerateSignal

    t_step      = 1/sampling_frequency;
    dot         = [t_min:t_step:t_max];
    disc_signal = GenerateSignal(t_min, t_max, t_step, nu);

    hold on; grid on;
    plot(dot, disc_signal, '.m');
    title('Time domain, sampling frequency ', sampling_frequency);
    legend('original signal', 'sampled signal', 'sampled dot')
end
```

Для нахождения спектра дискретного сигнала будем использовать самостоятельно реализованную в MATLAB функцию дискретного Фурье-преобразования (реализация работает для сигналов, у которых отсчеты расположены на положительной полуоси):

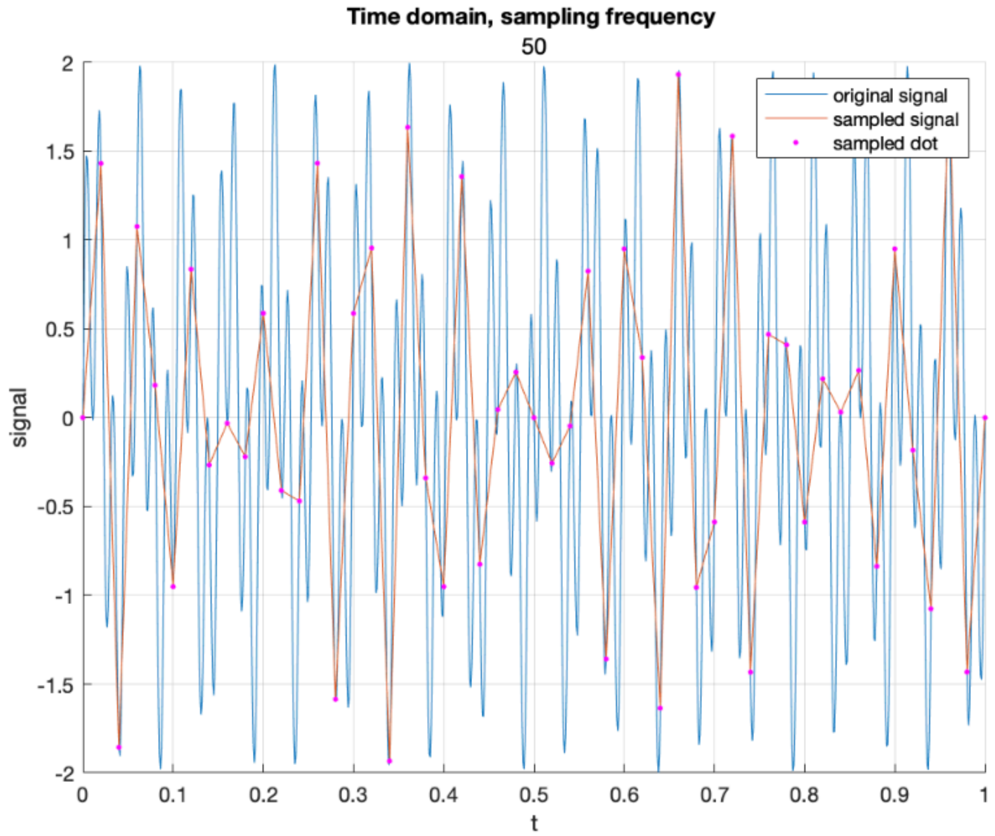
```
function frequency_domain = FourierTransformSamp(time_domain, sampling_frequency, F_max)
%Производит преобразование Фурье для дискретного сигнала, заданного набором
%отсчетов во временной области.

    t_step           = 1/sampling_frequency;
    nu               = [-F_max-10: 2*F_max/1000 :F_max+10];
    frequency_domain = zeros(1,length(nu));

    for k = 1:length(time_domain)
        frequency_domain = frequency_domain + time_domain(k)*exp(-2*pi.*nu*(k-
1)*t_step*j)*t_step;
    end

    %АЧХ
    hold on; grid on;
    plot(nu,abs(frequency_domain))
    xlabel('\nu'); ylabel('|S(\nu)|');
    title('Frequency domain')
end
```

В main-файле приводим реализацию использования данной функции при частотах дискретизации: 50, 135 и 200.

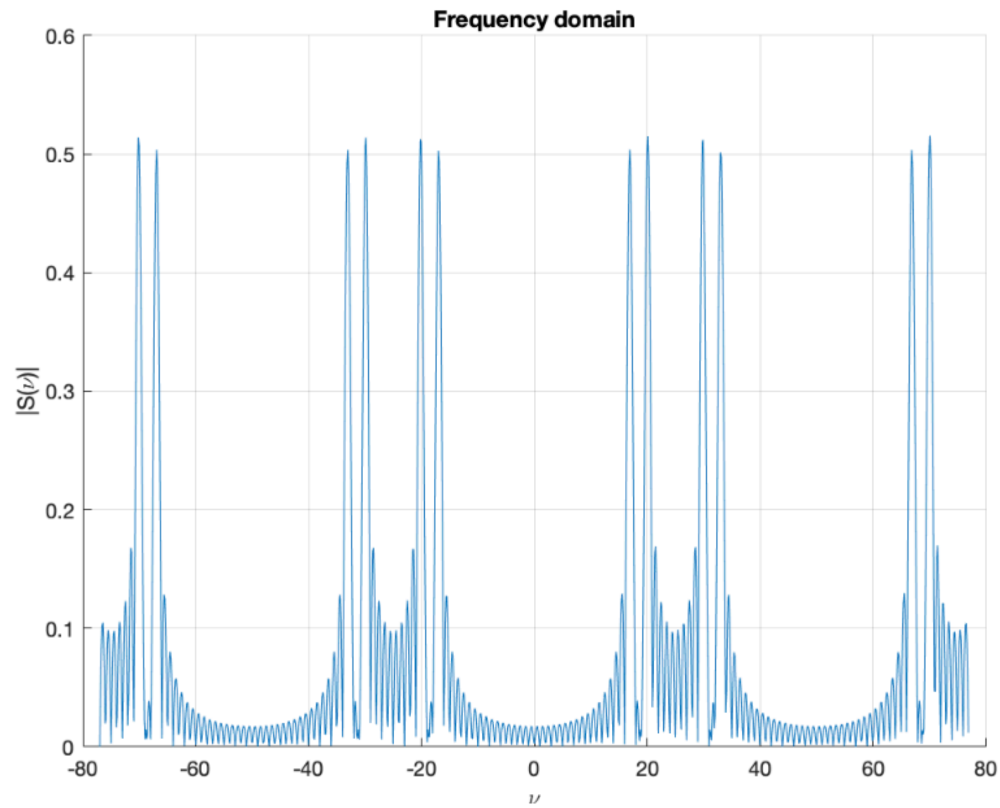
```
%TASK-1-PART-3-----------------------------------------
figure(2);
GenerateSignal(t_min,t_max,t_step,nu);
[dot_50, samp_signal_50] = SignalSamp(50,t_min,t_max,nu);
figure(3)
FourierTransformSamp(samp_signal_50, 50, max(nu(2),nu(1)));

figure(4);
```

```
GenerateSignal(t_min,t_max,t_step,nu);
[dot_135, samp_signal_135] = SignalSamp(135,t_min,t_max,nu);
figure(5)
FourierTransformSamp(samp_signal_135, 135, max(nu(2),nu(1)));

figure(6);
GenerateSignal(t_min,t_max,t_step,nu);
[dot_200, samp_signal_200] = SignalSamp(200,t_min,t_max,nu);
figure(7)
FourierTransformSamp(samp_signal_200, 200, max(nu(2),nu(1)));
```

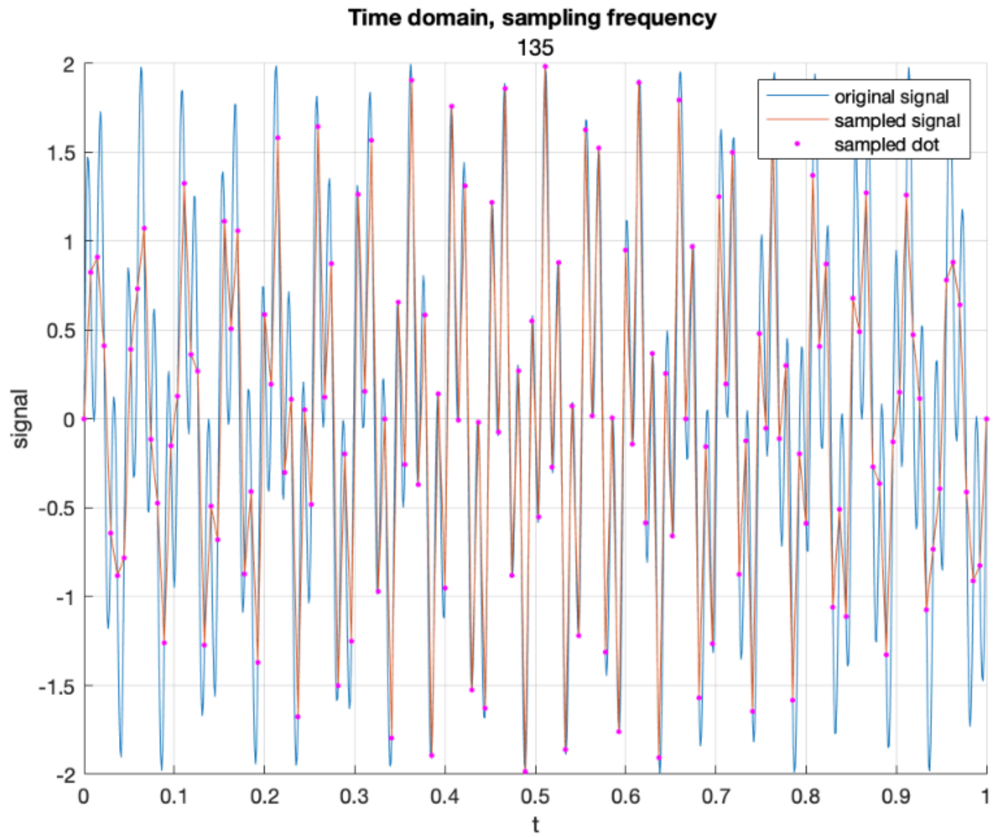
Получаем результат:

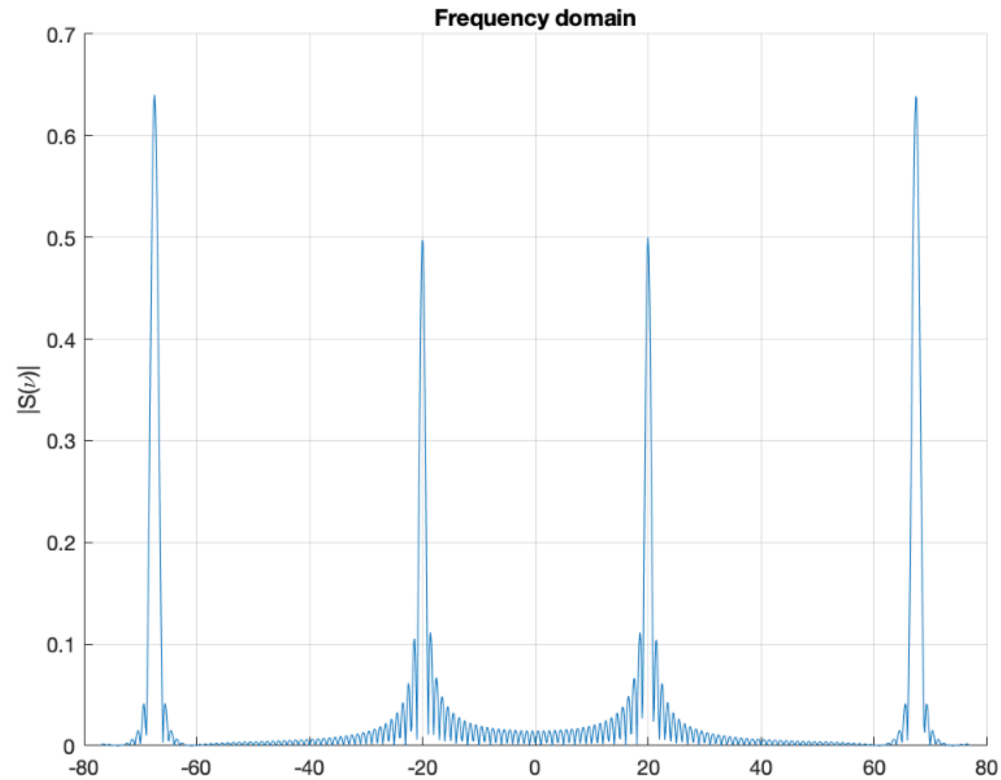


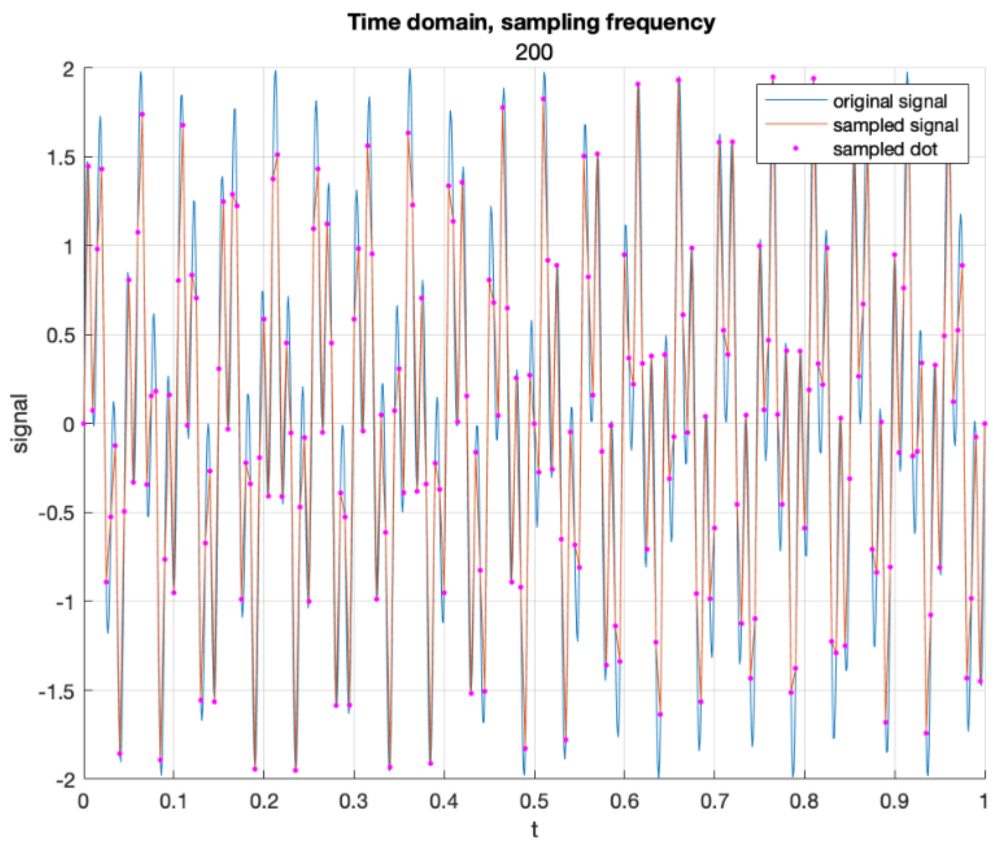
Time domain, sampling frequency
50
original signal
sampled signal
sampled dot
signal
t

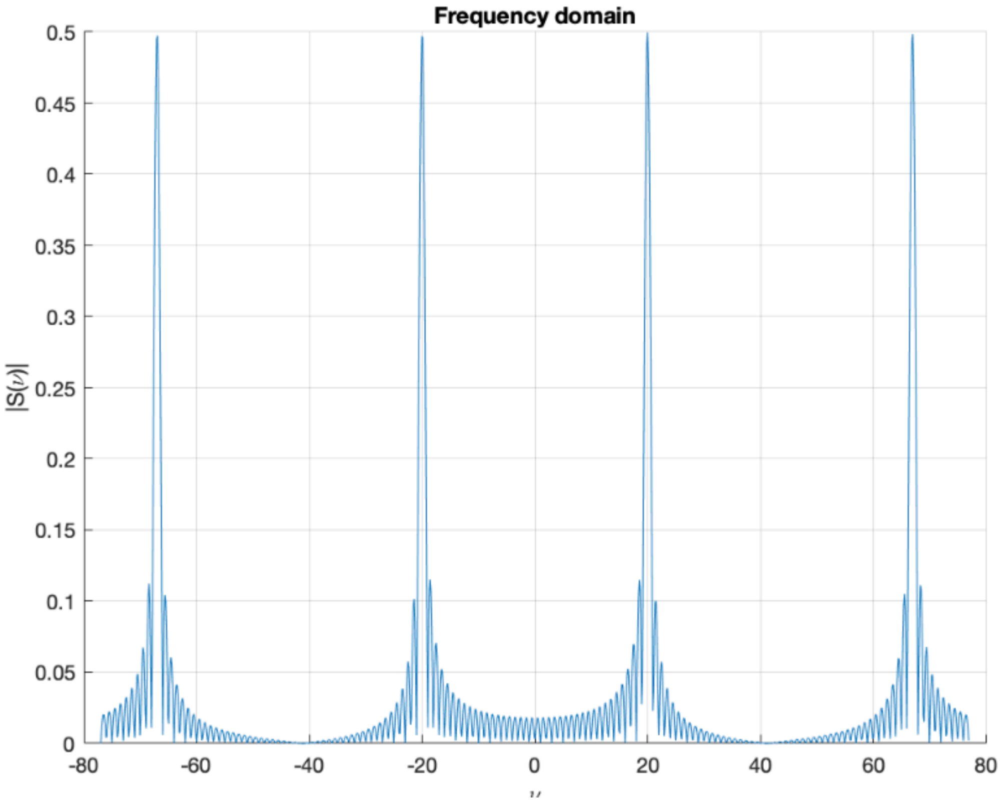


Frequency domain
|S(ν)|
ν


Time domain, sampling frequency
135
original signal
sampled signal
sampled dot
signal
t


Frequency domain
|S(ν)|





Вывод

При дискретизации сигнала важно было выбрать частоту дискретизации в соответствии с частотным критерием, после чего было выполнено построение при 3х различных частотах дискретизации - 50, 135, 200. При это частота дискретизации 50 не удовлетворяла частотному критерию, 135 удовлетворяла, но находилась почти на границе, а 200 удовлетворяла с излишком.

В этих условиях наблюдаем в чатотной области (а именно на графике амплитуды частотного спектра) 12 пиков, соотвествующих частоте дискретизации 50 и 4 пика, соответствующих частотам 135 и 200.

- На первых 2х изображениях (для чатоты 50) может наблюдать эффект наложения частот и подобный частотный спектр могут иметь несколько различных сигналов, дискретизации которых совпадают. От этого мы видим несколько значений частот, которые мог бы иметь сигнал с такими свойствами. Что конечно не отражает реальной картины.
- На последних 4х изрбражениях (для частот 135 и 200) имеем пики в значениях $\pm 20, \pm 67$, что в точночти соотвествует частотам синусов ν_1, ν_2, входящих в исследуемый сигнал. Таким образом, удовлетворив частотному критерию выбора шага дискретизации мы получаем картину в точности соотвествующую реальности и очень хорошо описывающую исследуемый сигнал.

4. Иллюстрация эффекта наложения частот

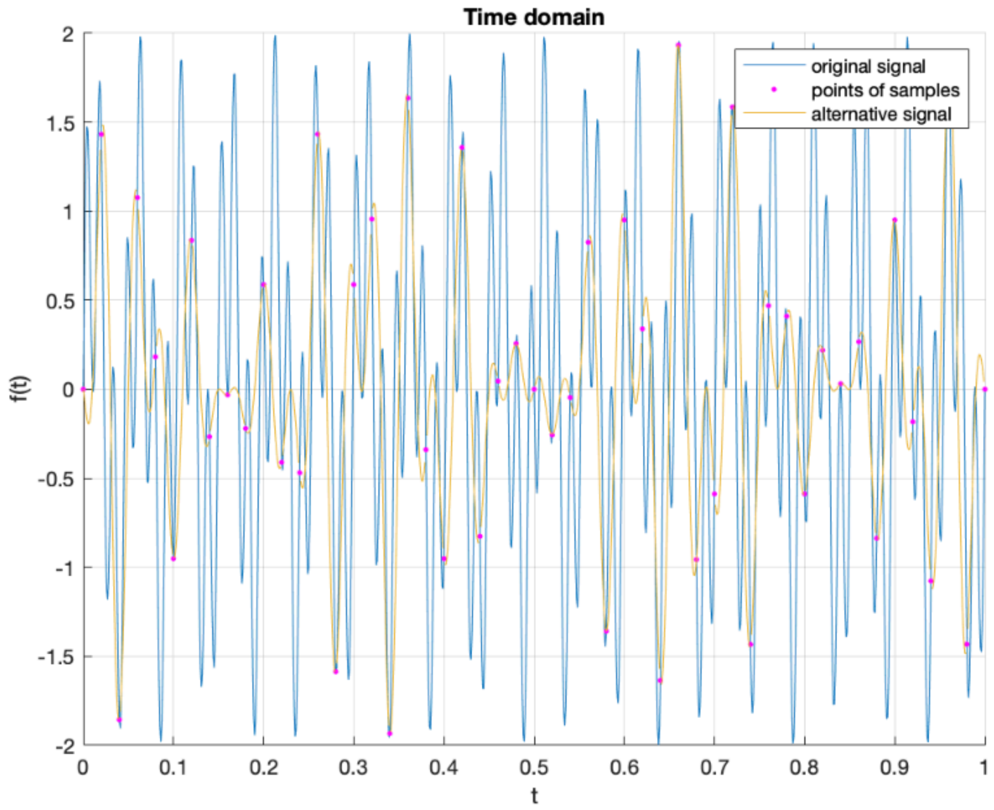
Для иллюстрации эффекта наложения частот была реализована и использована функция построения ряда Котельникова по набору отсчетов.

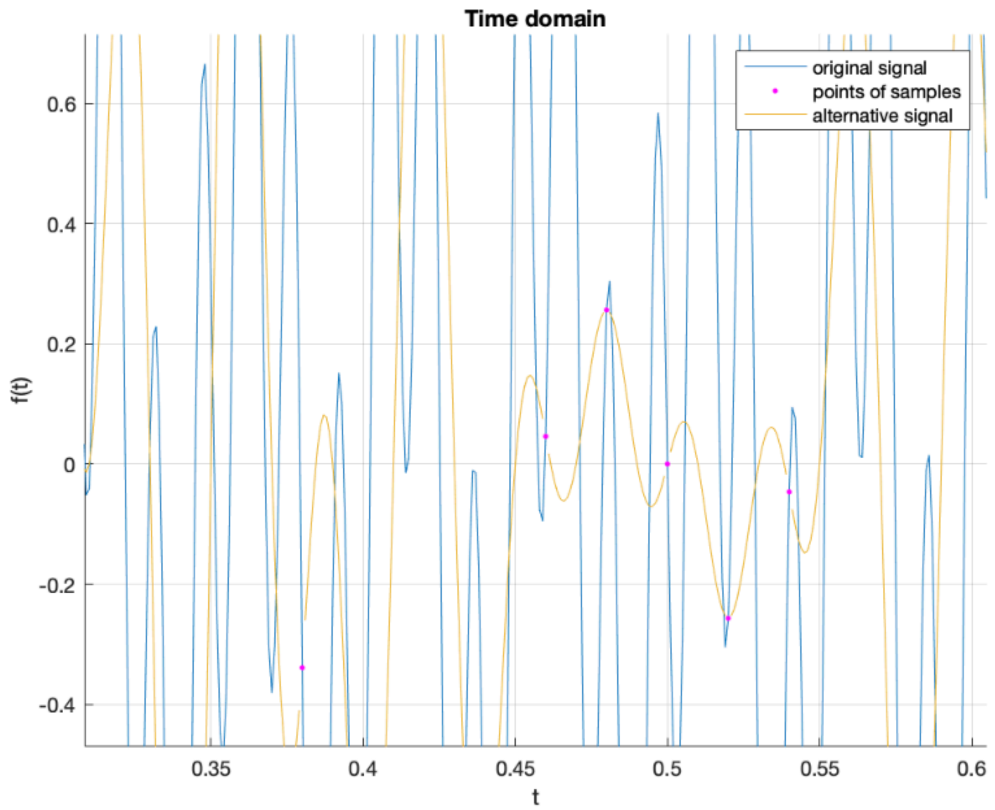
```
function series = KotelnikovSeries(set_of_samples, t_min, t_max, original_step,
sampling_frequency)
%Функция восстанавливает сигнал по набору его отсчетов при помощи теоремы
%Котельникова (предполагает предварительное построение графиков исходного
%сигнала до дискретизации и набора отсчетов внешним образом)
    syms t

    t_step = 1/sampling_frequency;
    kt     = [t_min: t_step: t_max];
    KotelnikovSer = matlabFunction(sum(set_of_samples.*(sin(2*pi*sampling_frequency*(t-
kt))./((2*pi*sampling_frequency*(t-kt)))));

    t      = [t_min: original_step :t_max];
    series = KotelnikovSer(t);

    hold on; grid on;
    plot(t,series);
    title('Time domain')
    xlabel('t'); ylabel('f(t)');
    legend('original signal', 'points of samples', 'alternative signal')
end
```

В main-файле применения этой функции к ншим условиям для частоты дискретизации 50 (ниже допустимой) выглядит следующим образом.

```
figure(8)
hold on; grid on;
plot([t_min: t_step: t_max], signal);
plot(dot_50, samp_signal_50, '.m')
alternative_signal = KotelnikovSeries(samp_signal_50, t_min, t_max, t_step, 50);
```

Получаем результат (2 рисунка в разном масштабе) - 2 сигнала: исходный и полученный при восстановлении с ряда Котельникова.





Видим, что в этом случае, вообще говоря, сигналы различны, однако их дискретизации совпадают полностью, и, как следствие, их дискретные спектры. Это и является иллюстрацией эффекта наложения частот, который произошел из-за недопустимо низкой частоты дискретизации сигнала.

5. Обработка изображений.

Реализация алгоритма простого выкалывания из матрицы изображения :

```
function output_image = BruteForceGouging(input_image, k)
%Грубое уменьшение частоты дискретизации изображения в n раз и вывод
%полученного изображения.

    output_image = zeros(floor(size(input_image,1)/k), floor(size(input_image,2)/k), 3,
'uint8');
    for i=1:size(output_image,1)
        for j=1:size(output_image,2)
            output_image(i, j, :) = input_image(floor(i*k), floor(j*k), :);
        end
    end
    imshow(output_image)
end
```

Применим:

```
image = imread('var1.png');

figure(10)
BruteForceGouging(image, 2);
im2 = imresize(image, 1./2);
figure(11)
imshow(im2)

figure(12)
BruteForceGouging(image, 3);
im3 = imresize(image, 1./3);
figure(13)
imshow(im3)

figure(14)
BruteForceGouging(image, 4);
im4 = imresize(image, 1./4);
figure(15)
imshow(im4)
```

Имеем результат обработки:

Слева - грубое выкалывание, справа - рещультат функции imresize().

Ясно видно, что поскольку при использовании функции imresize происходит дополнительная интерполяция в клетки матрицы, то изображения выглядят куда лучше грубого выбрасывания.

Часть 2. Исследование эффектов квантования.

Цель работы

Исследование влияния параметров квантования на качество сигналов, изучение статистических аспектов квантования.

Ход работы:

1. Синтезируем случайный дискретный сигнал $x(k)$, состоящий из нескольких сотен отсчетов с одинаковым равномерным распределением. Построим по отсчетам его график.

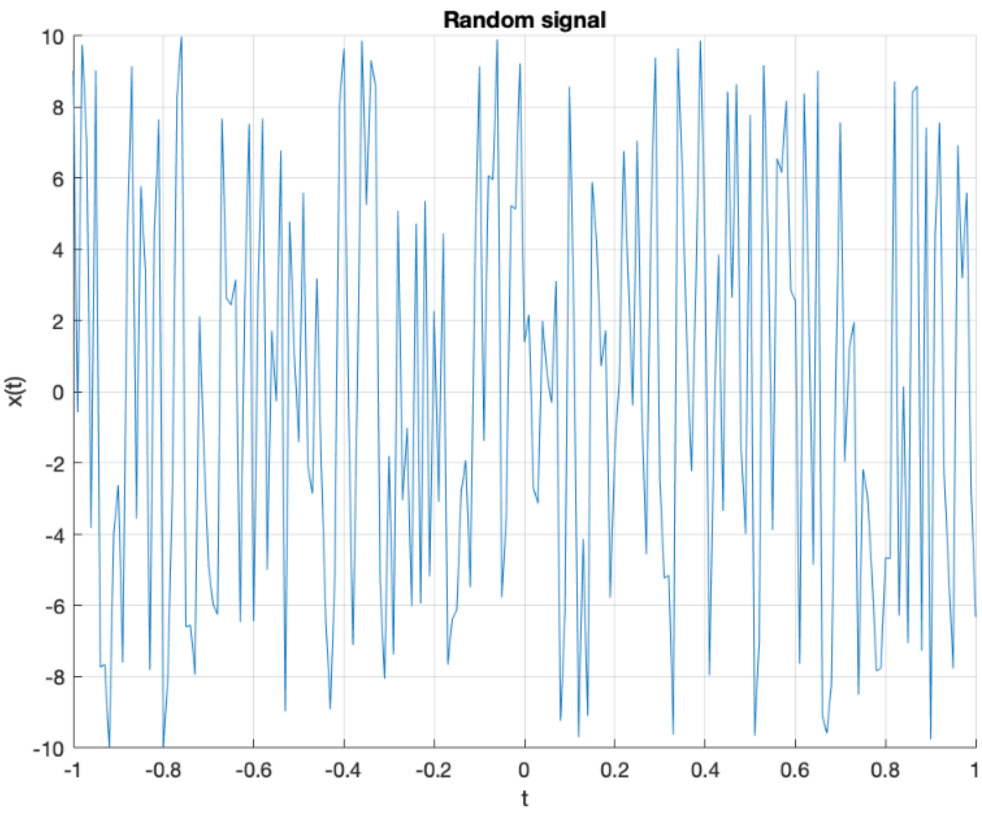
Для генерации сигнала была использована функция:

```
function signal = GenerateRandomSignal(count_set, x_min, x_max)
%Генерация случайного равномерно распределенного на отрезке [x_min, x_max]
%сигнала, заданного на отрезке count_set + построение графика.

    signal = unifrnd(x_min,x_max,1,length(count_set));

    hold on;grid on;
    plot(count_set,signal);
    title('Random signal');
    xlabel('t'); ylabel('x(t)');
end
```

Реализация использования данной функции выглядит следующим образом:

```
%TASK-2-PART-1-------------------------------------------
count_set = [-1:0.01:1];
x_min     = -10;
x_max     = 10;

figure(1)
original_signal = GenerateRandomSignal(count_set, x_min, x_max);
```

Получаем результат:



2. Проведем равномерное квантование данного сигнала.

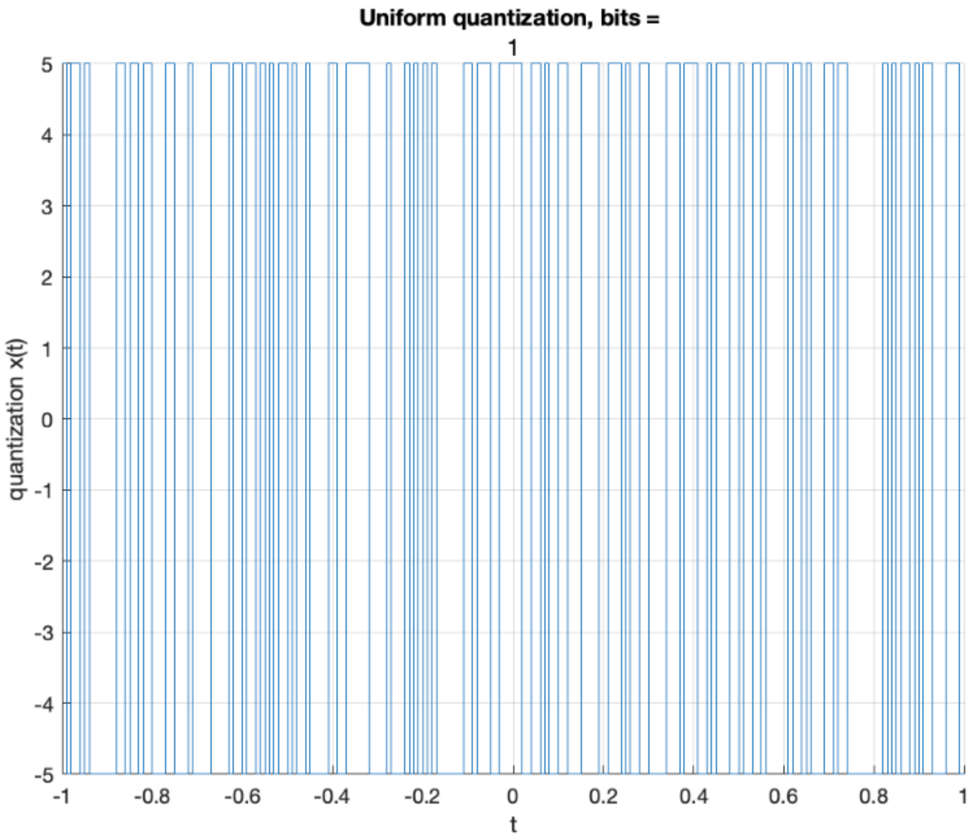
Функция равномерного квантования сигнала на k бит выглядит реализована следующим образом:

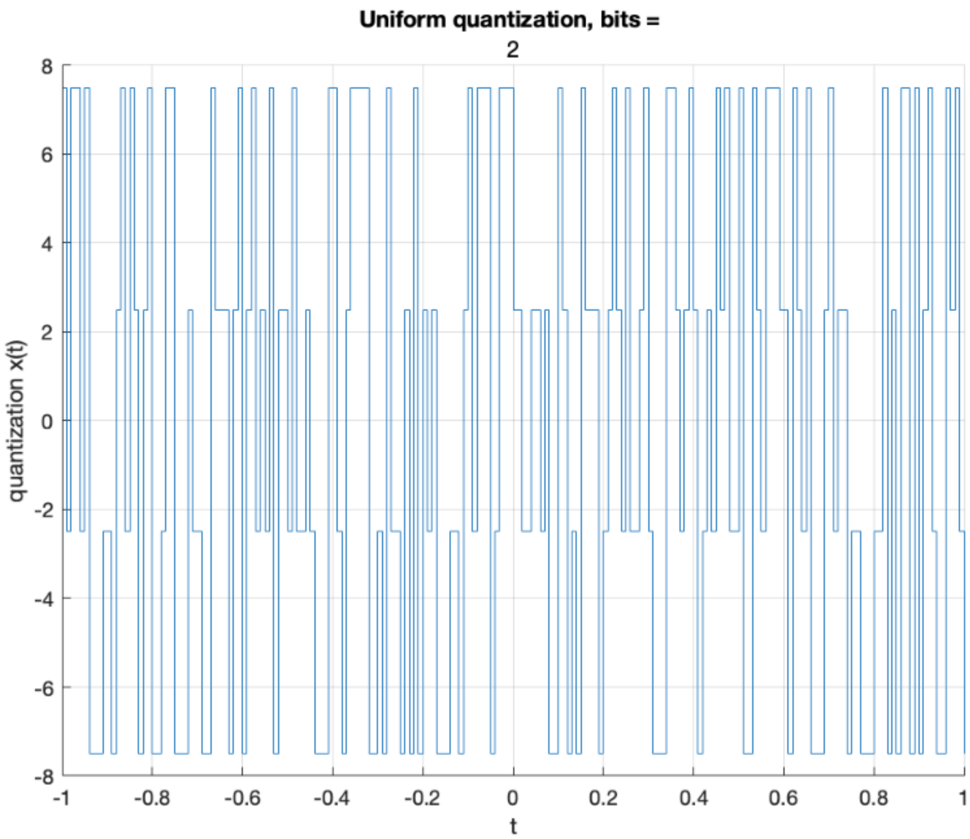
```
function output_signal = UniformQuantization(original_signal, count_set, bits, x_min, x_max)
%Функция равноерного квантования дискретного сигнала и построение графика
%квантованного сигнала

        N                    = 2^bits;
        q_step               = (x_max-x_min)/N;
        output_signal        = zeros(1,length(count_set));

        for i=1:length(original_signal)
            l = floor(original_signal(i)./q_step);
            if ((mod(original_signal(i), q_step) == 0) && (l > 0))
                l = l - 1;
            end
            output_signal(i) = q_step.*(2.*l + 1)./2;
        end

        hold on; grid on;
        stairs(count_set, output_signal);
        title('Uniform quantization, bits = ', bits);
        xlabel('t'); ylabel('quantization x(t)');
end
```

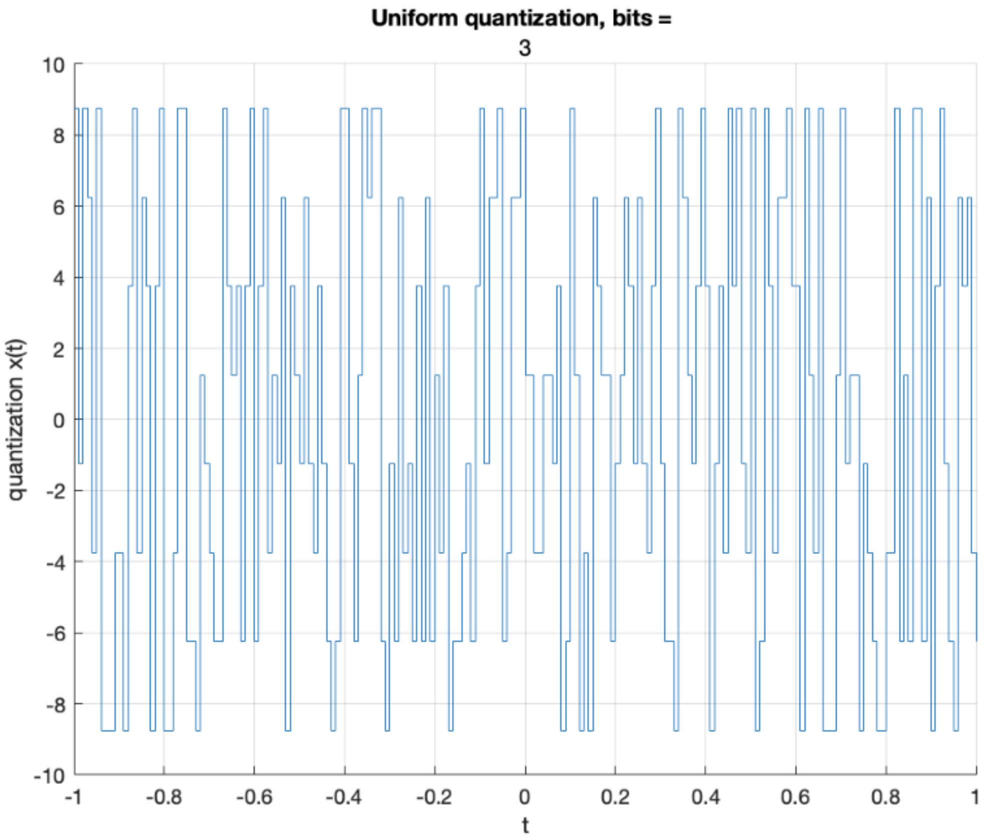
Ее применение:

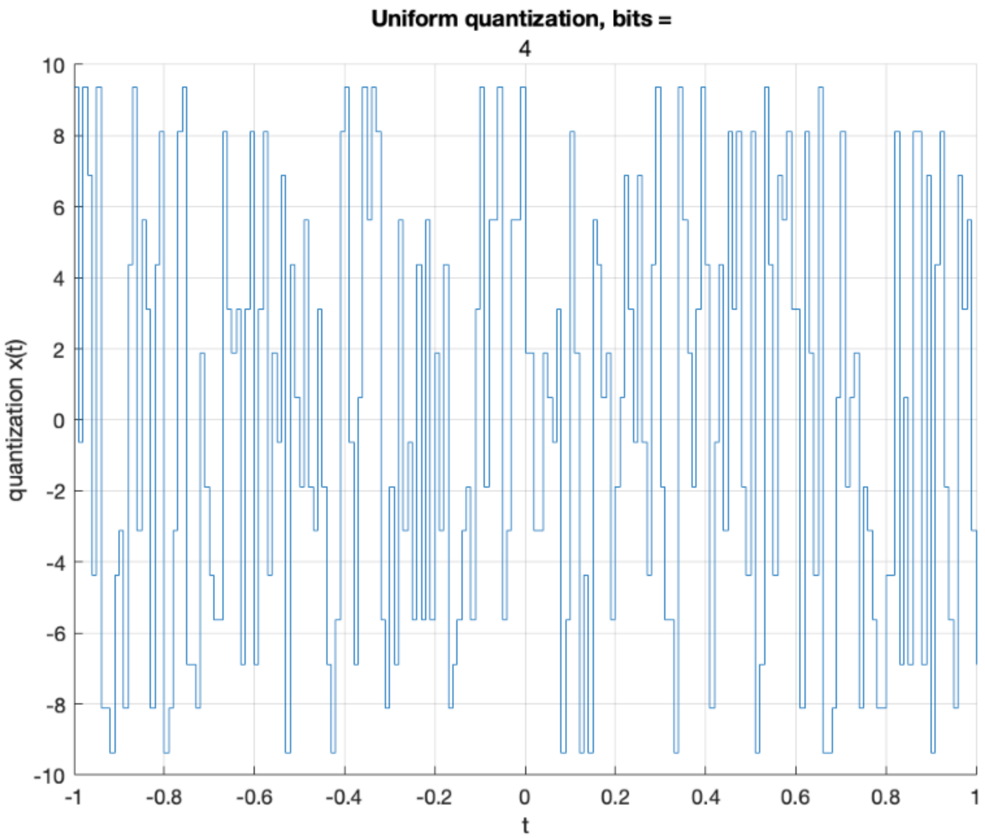
```
quantorization_signal = zeros(length(original_signal), 8);
for k = 1:1:8
    figure(k+1)
    quantorization_signal(:,k) = UniformQuantization(original_signal, count_set, k, x_min,
x_max);
end
```

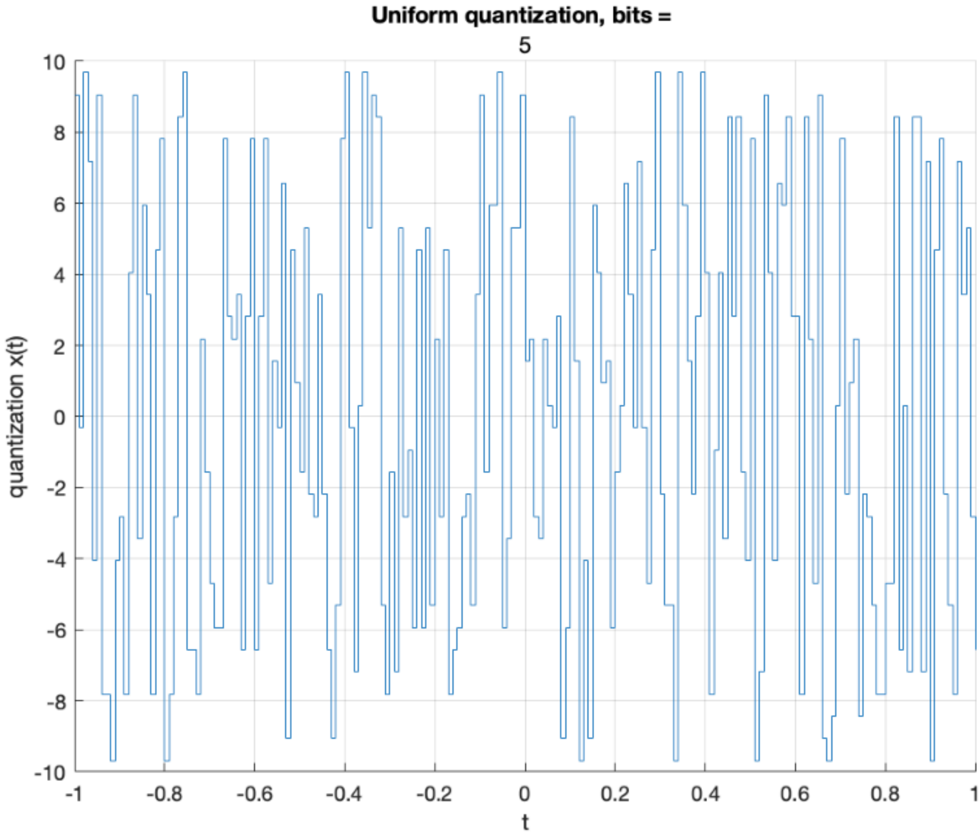
Результат работы:

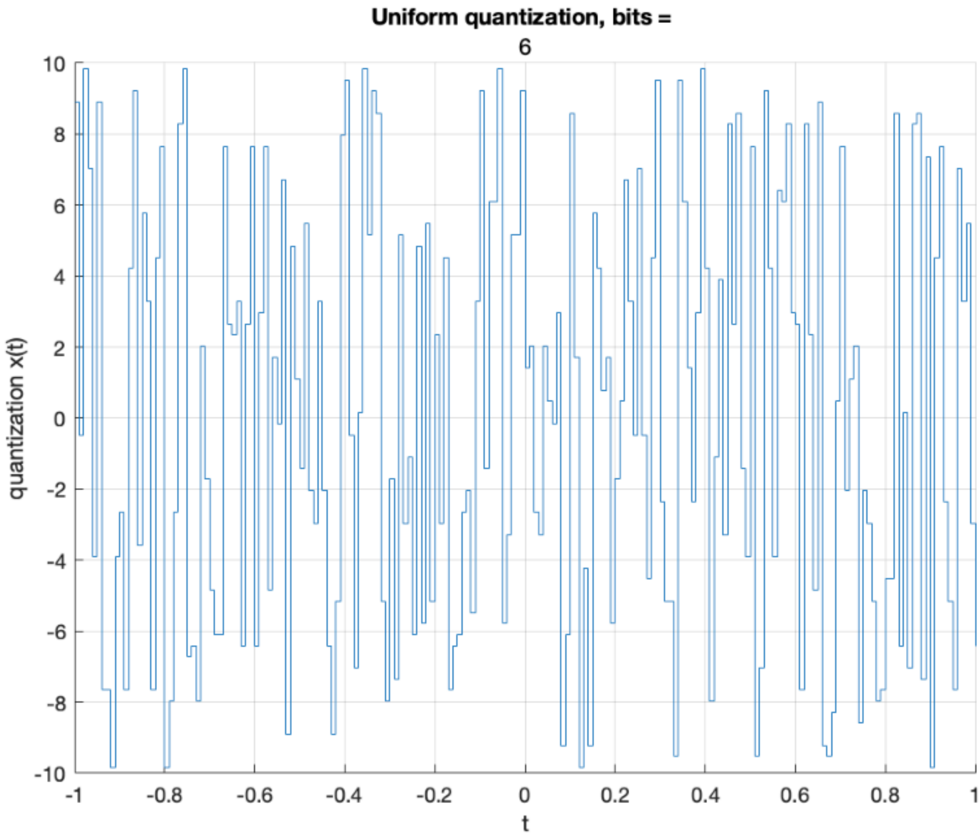

Uniform quantization, bits =
1
quantization x(t)
t
5
4
3
2
1
0
-1
-2
-3
-4
-5
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

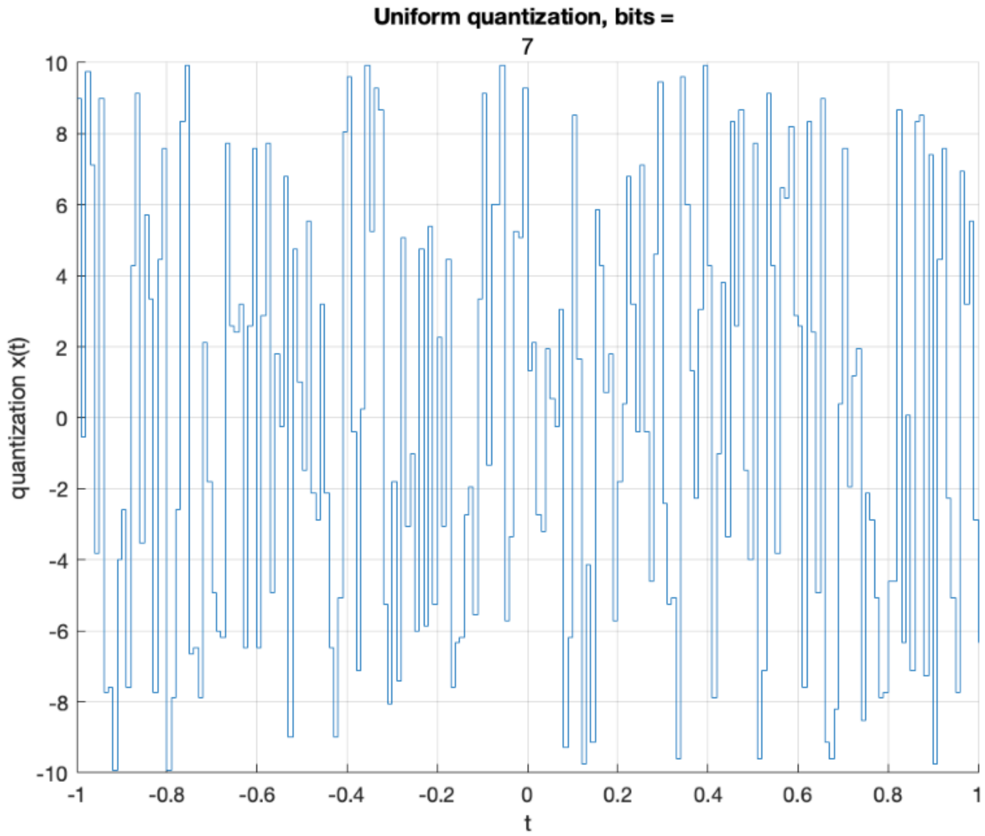

Uniform quantization, bits =
2
quantization x(t)
t
8
6
4
2
0
-2
-4
-6
-8
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

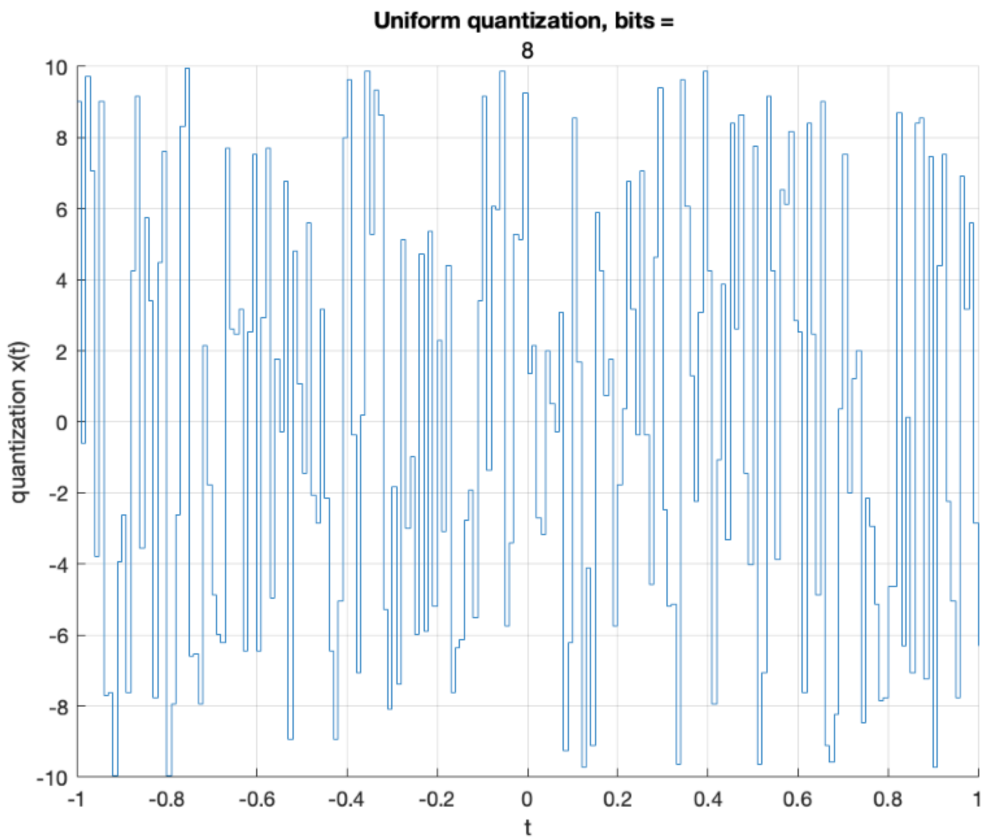

Uniform quantization, bits =
3
quantization x(t)
t
10
8
6
4
2
0
-2
-4
-6
-8
-10
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Uniform quantization, bits =
4
quantization x(t)
t
10
8
6
4
2
0
-2
-4
-6
-8
-10
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Uniform quantization, bits =
5
quantization x(t)
t
10
8
6
4
2
0
-2
-4
-6
-8
-10
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Uniform quantization, bits =
6
quantization x(t)
t
10
8
6
4
2
0
-2
-4
-6
-8
-10
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Uniform quantization, bits =
7
quantization x(t)
t


Uniform quantization, bits =
8
quantization x(t)
t

3. Эксперементальная оценка ошибки квантования

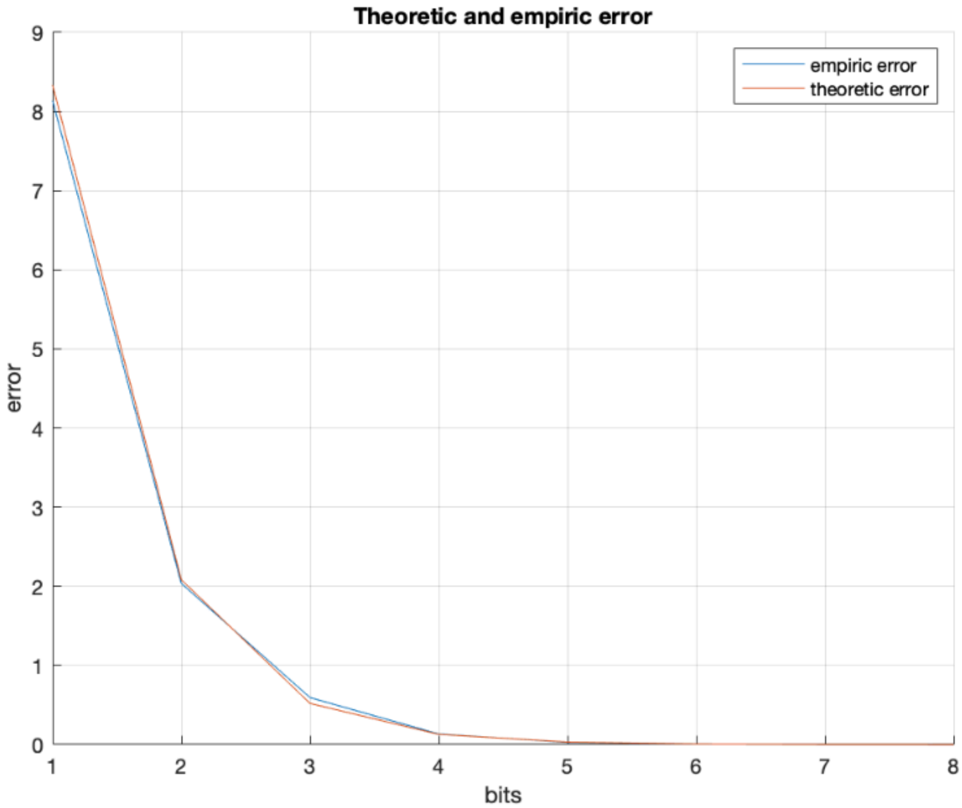
Реализуем функцию, анализирующую квантованый и оригинальный сигналы и оценивающую ошибку квантования:

```
function  [error_empiric, error_theoretic] = QuantizationError(original_signal,
quantorization_signal, x_min, x_max)
%Анализ ошибки квантования: вычисление эмпирической и теоретической ошибок
%равномерного квантования, построение графиков

    error_empiric   = zeros(1,length(quantorization_signal(:,1)));
    error_theoretic = zeros(1,length(quantorization_signal(:,1)));

    for k = 1:1:length(quantorization_signal(:,1))
        error_empiric(k)   = sum((original_signal-
quantorization_signal(k,:)).^2)./length(original_signal);
        error_theoretic(k) = ((x_max-x_min)/2^k)^2/12;
    end

    hold on; grid on;
    plot([1:length(error_empiric)], error_empiric);
    plot([1:length(error_theoretic)], error_theoretic);
    xlabel('bits'); ylabel('error');
    title('Theoretic and empiric error');
    legend('empiric error', 'theoretic error');
end
```

Применение этой функции к нашей задаче:

```
figure(10)
[error_empiric, error_theoretic] = QuantizationError(original_signal, quantorization_signal,
x_min, x_max)
```

Получаем результат:




```
error_empiric = 1×8
    8.1375    2.0372    0.5944    0.1339    0.0290    0.0080    0.0021    0.0006

error_theoretic = 1×8
    8.3333    2.0833    0.5208    0.1302    0.0326    0.0081    0.0020    0.0005
```

Вывод Теоретическая и эмпирическая ошибки почти не отличимы друг от друга, при разном количестве бит, используемых для квантования сигнала доминирующее положение в точности чередуется. Что касается зависимости от количесвта используемыз бит, то ясно видно, что с ростом этого количества величина ошибки уменьшается.

4. Анализ SNR ошибки квантования

Аналогичным образом напишем функцию, но использующую другую меру ошибки квантования:

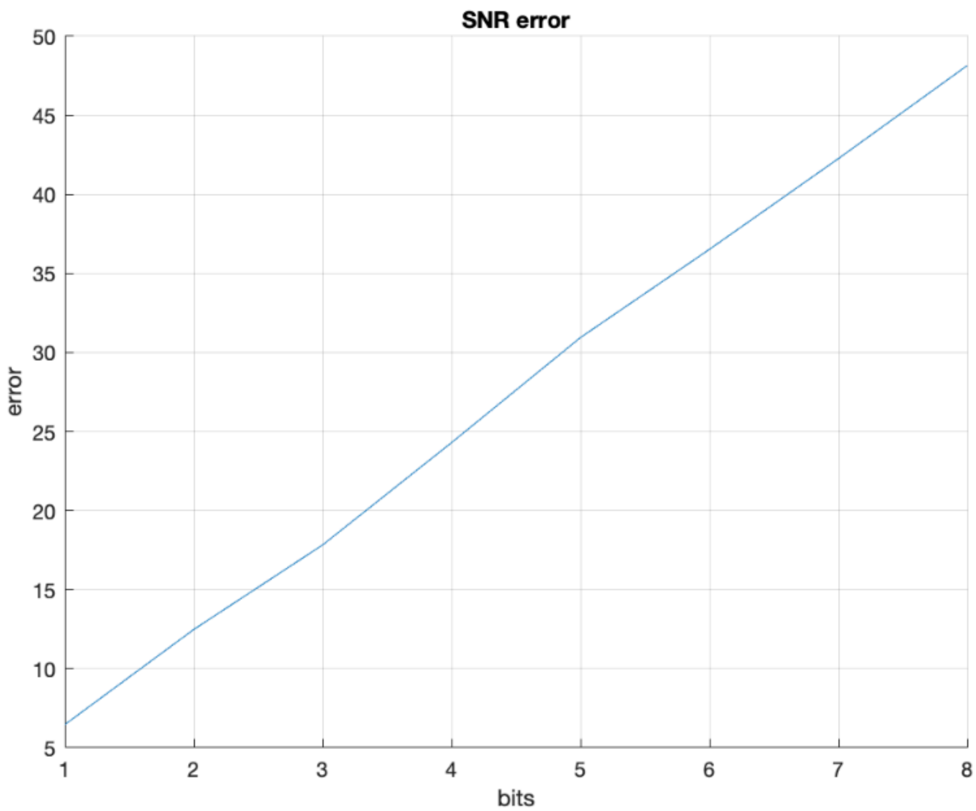
```
function  error_empiric = QuantizationErrorSNR(original_signal, quantorization_signal, x_min, x_max)
%Анализ ошибки квантования: вычисление SNR ошибки
%равномерного квантования, построение графика

    error_empiric   = zeros(1,length(quantorization_signal(:,1)));

    for k = 1:1:length(quantorization_signal(:,1))
        A_signal           = sqrt(sum(original_signal.^2))/length(original_signal);
        A_noise            = sqrt(sum((original_signal-quantorization_signal(k,:)).^2))/length(original_signal);
        error_empiric(k)    = 20*log10(A_signal/A_noise);
        error_theoretic(k) = ((x_max-x_min)/2^k)^2/12;
```

```
    end

    hold on; grid on;
    plot([1:length(error_empiric)], error_empiric);
    xlabel('bits'); ylabel('error');
    title('SNR error');
end
```

Применения к нашему моделированию:

```
figure(11)
error_SNR = QuantizationErrorSNR(original_signal, quantorization_signal, x_min, x_max)
```

Итог:




```
error_SNR = 1×8
    6.4673    12.4818    17.8315    24.3029    30.9486    36.5258    42.2634    48.1535
```

Вывод: Наблюдаем похожую ситуацию, с увеличеснием количества бит квантования шибки уменьшается. Однако, куда интереснее то, что график имеет линейный вид, это позволяет судить о скорости убывания ошибки квантования, а именно - она убывает как $O(bits^{10})$.

5. Нормальный дискретный случайный сигнал

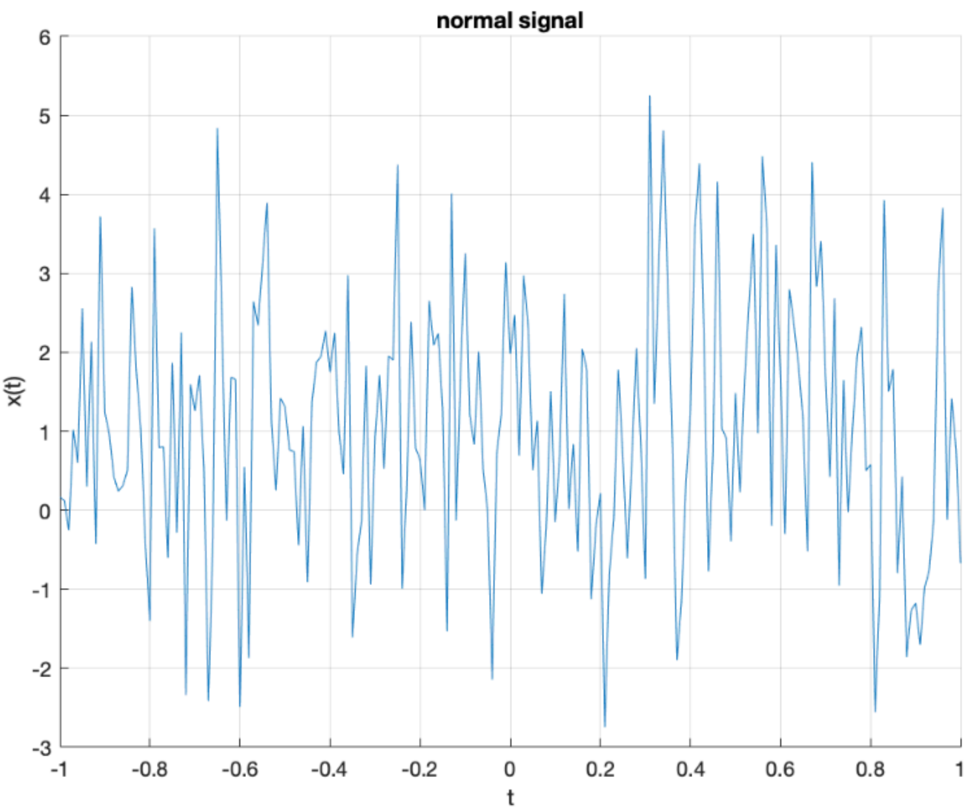
Функция генерации выглядит следующим образом:

```
function output_signal = GenerateRandomNormalSignal(m, sigma, count_set)
%Генерация случайного дискретного равномерно распределенного случайного
%сигнала.

    output_signal = m + sqrt(sigma).*randn(1, length(count_set));

    hold on; grid on;
    plot(count_set, output_signal);
    xlabel('t'); ylabel('x(t)');
    title('normal signal');
end
```

Собственно генерация:

```
m = 1; sigma = 2;
figure(12)
normal_signal = GenerateRandomNormalSignal(m, sigma, count_set);
```

Результат:



6. Параметры оптимального квантоватеня Ллойда-Макса

Опишем функцию, способную вычислять параметры квантователя Ллойда-Макса для произвольного нормального распределения:

```
function [t_new, d_new] = LloidMaksForNormal(t, d, m, sigma)
%Вычисляет параметры квантователя Ллоида-Макса для нормального
%распределения с произвольными параметрами
    t_new = zeros(1,length(t));
```

```
    d_new = zeros(1,length(d));

    for k = 1:length(t)
        t_new(k) = t(k)*sigma + m;
    end

    for k = 1:length(d)
        d_new(k) = d(k)*sigma + m;
    end
end
```

Применим в нашем моделировании:

```
% При использовании 1 бита на отсчет (2 уровня):
t1 = [-inf 0 inf];
d1 = [-0.7979 0.7979];
% При использовании 2 бит на отсчет(4 уровня):
t2 = [-inf -0.9816 0 0.9816 inf];
d2 = [-1.5104 -0.4528 0.4528 1.5104];
% При использовании 3 бит на отсчет (8 уровней):
t3 = [-inf -1.7479 -1.05 -0.5005 0 0.5005 1.05 1.7479 inf];
d3 = [-2.1519 -1.3439 -0.756 -0.2451 0.2451 0.756 1.3439 2.1519];
% При использовании 4 бит на отсчет (16 уровней):
t4 = [-inf -2.4008 -1.8435 -1.4371 -1.0993 -0.7995 -0.5224 -0.2582 0 0.2582 0.5224 0.7995 1.0993
1.4371 1.8435 2.4008 inf];
d4 = [-2.7326 -2.069 -1.618 -1.2562 -0.9423 -0.6568 -0.3880 -0.1284 0.1284 0.3880 0.6568 0.9423
1.2562 1.618 2.069 2.7326];

[t1_new,d1_new] = LloidMaksForNormal(t1,d1,m,sigma)
[t2_new,d2_new] = LloidMaksForNormal(t2,d2,m,sigma)
[t3_new,d3_new] = LloidMaksForNormal(t3,d3,m,sigma)
[t4_new,d4_new] = LloidMaksForNormal(t4,d4,m,sigma)
```

Получим результат:

```
t1_new = 1×3
  -Inf     1   Inf

d1_new = 1×2
   -0.5958    2.5958

t2_new = 1×5
      -Inf   -0.9632    1.0000    2.9632       Inf

d2_new = 1×4
   -2.0208    0.0944    1.9056    4.0208

t3_new = 1×9
      -Inf   -2.4958   -1.1000   -0.0010    1.0000    2.0010    3.1000    4.4958       Inf

d3_new = 1×8
   -3.3038   -1.6878   -0.5120    0.5098    1.4902    2.5120    3.6878    5.3038
```

```
t4_new = 1×17
      -Inf    -3.8016    -2.6870    -1.8742    -1.1986    -0.5990    -0.0448     0.4836     1.0000
 1.5164     2.0448     2.5990     3.1986     3.8742     4.6870     5.8016        Inf

d4_new = 1×16
   -4.4652    -3.1380    -2.2360    -1.5124    -0.8846    -0.3136     0.2240     0.7432     1.2568
 1.7760     2.3136     2.8846     3.5124     4.2360     5.1380     6.4652
```

7. Оптимальное квантование сигнала $x(k)$, используя от 1 до 4х битов на отсчет

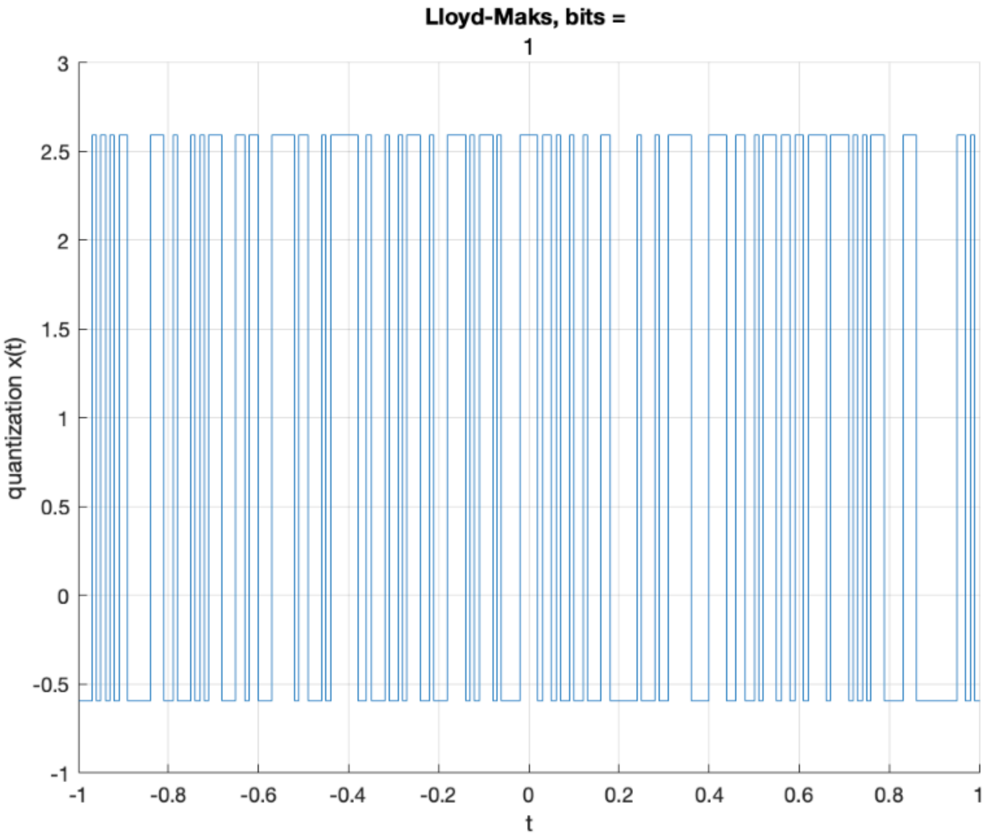
По-традиции организуем вычисления в виде m-функции:

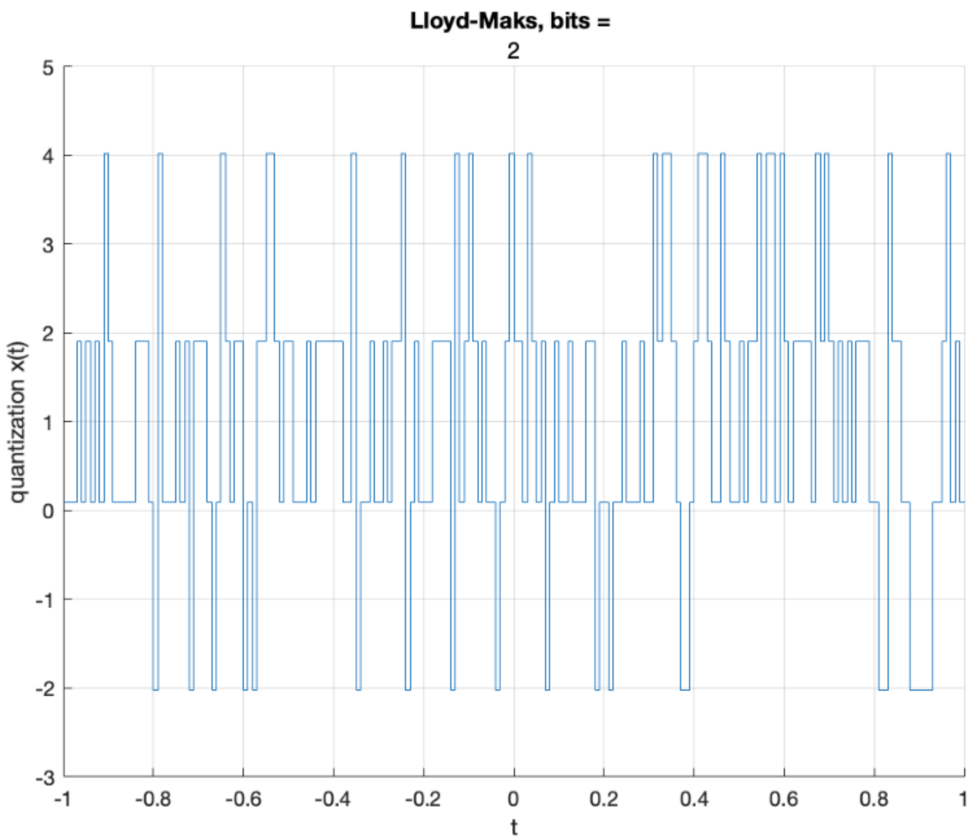
```
function quant = OptimalLloydMaxQuantizer(original_signal, t, d, count_set)
%Производит оптимальное квантование Ллойда-Макса с заданным числом битов на
%один отсчет, которое определяется в зависимости от передаваемых параметров
%квантователя

    len = length(original_signal);
    quant = zeros(1, len);
    for i=1:len
        v = sum(original_signal(i) > t);
        quant(i) = d(v);
    end

    hold on; grid on;
    stairs(count_set, quant)
    title('Lloyd-Maks, bits = ', log2(length(d)))
    xlabel('t'); ylabel('quantization x(t)')
end
```

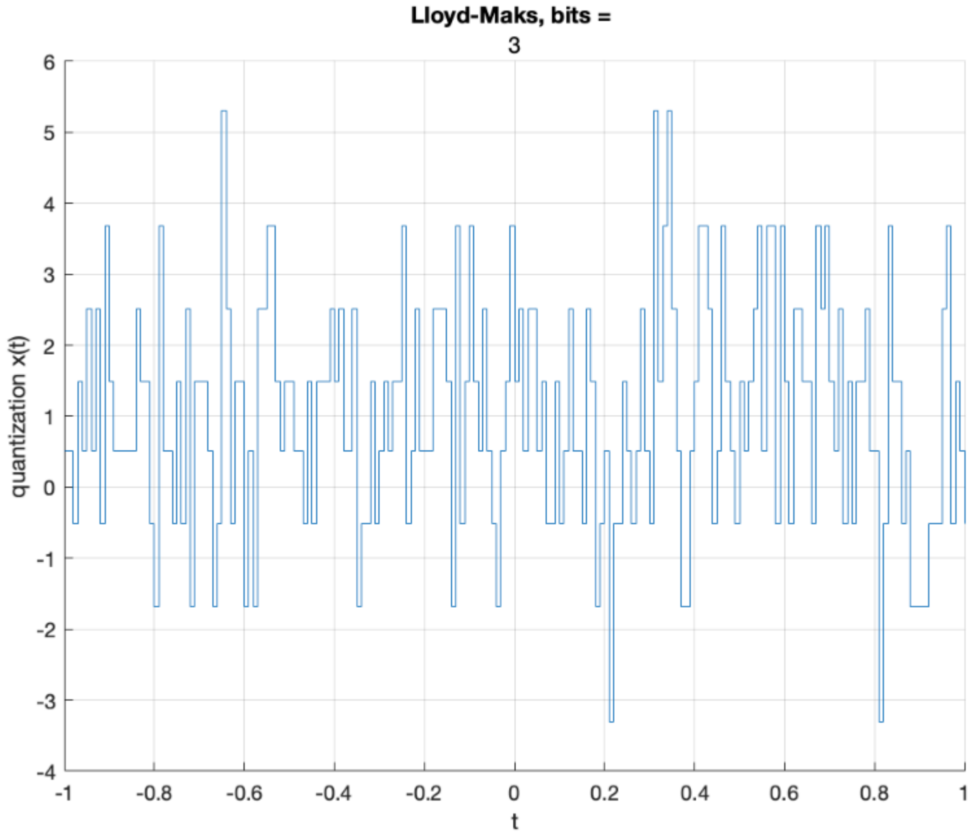
Далее применим к смоделированному сигналу:

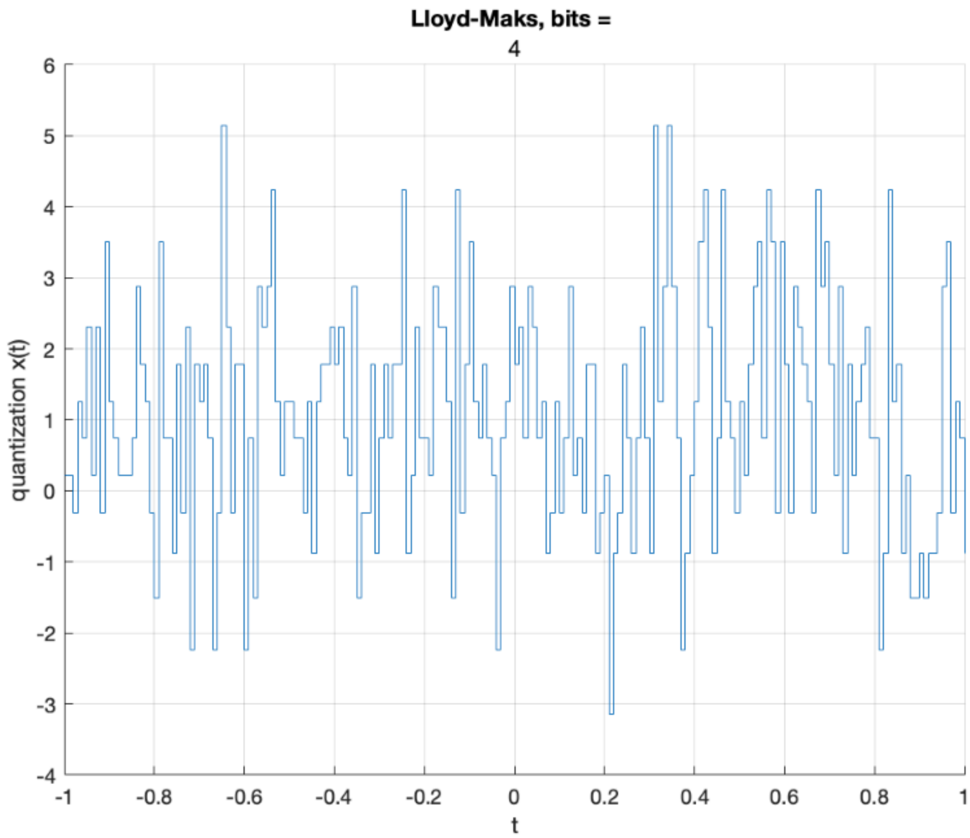
```
figure(13)
OptimalLloydMaxQuantizer(normal_signal, t1_new, d1_new, count_set);

figure(14)
OptimalLloydMaxQuantizer(normal_signal, t2_new, d2_new, count_set);

figure(15)
OptimalLloydMaxQuantizer(normal_signal, t3_new, d3_new, count_set);

figure(16)
OptimalLloydMaxQuantizer(normal_signal, t4_new, d4_new, count_set);
```

Получим результат:


Lloyd-Maks, bits =
1
quantization x(t)
t


Lloyd-Maks, bits =
2
quantization x(t)
t


Lloyd-Maks, bits =
3
quantization x(t)
t

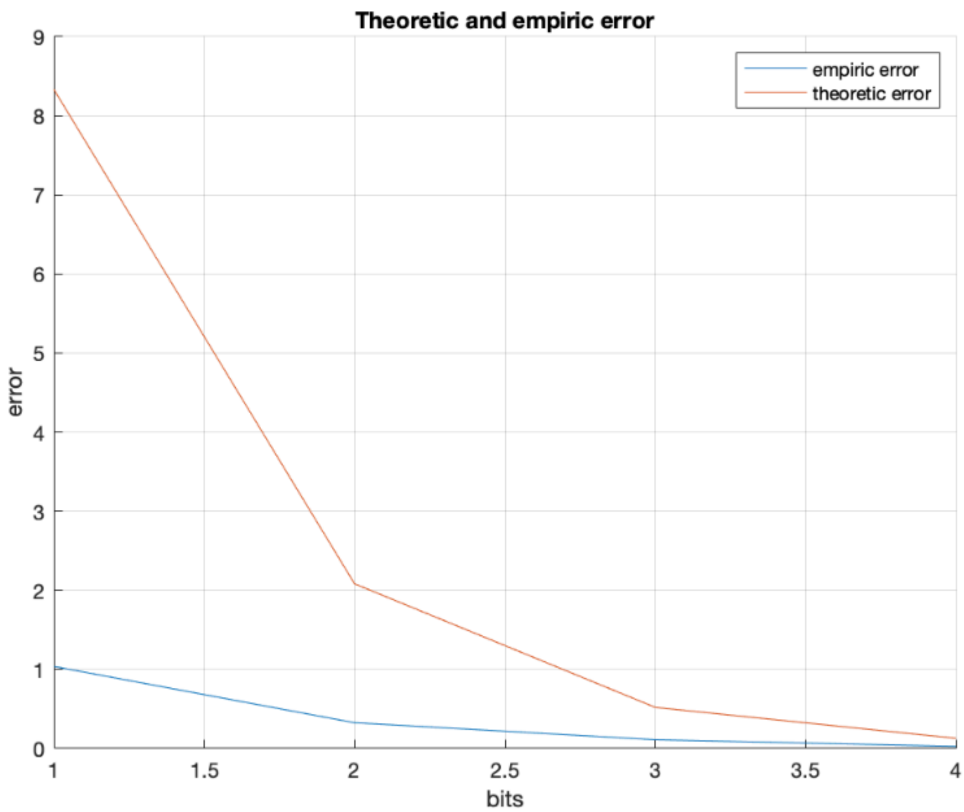

Lloyd-Maks, bits =
4
quantization x(t)
t

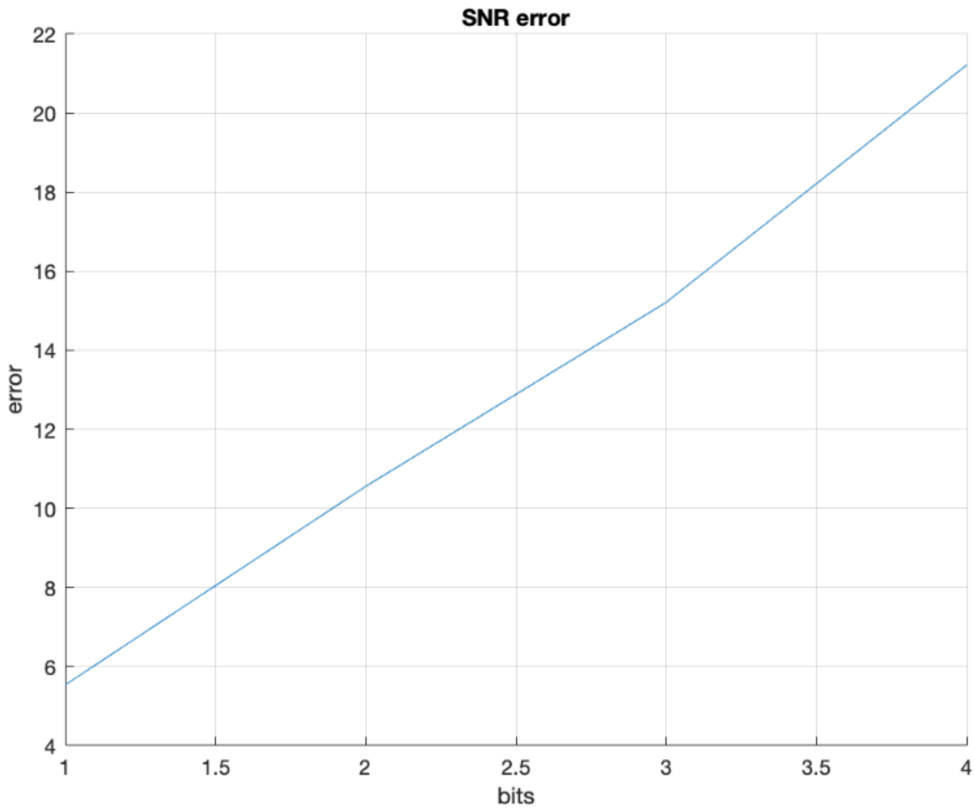
8. Определение ошибки оптимального квантования Ллойда-Макса

Используем уже реализованные функции анализа эмпирических и теоретических ошибок, а также SNR.

```
figure(17)
[error_empiric_norm, error_theoretic_norm] = QuantizationError(normal_signal,
quantorization_signal_norm, x_min, x_max)

figure(18)
error_SNR_norm = QuantizationErrorSNR(normal_signal, quantorization_signal_norm, x_min, x_max)
```

Имеем:






```
error_empiric_norm = 1×4
    1.0386    0.3267    0.1118    0.0281

error_theoretic_norm = 1×4
    8.3333    2.0833    0.5208    0.1302

error_SNR_norm = 1×4
    5.5353   10.5582   15.2150   21.2175
```

Вывод:

По первому графику можно заметить, что при маленьеом числе битов, использованных при квантовании эмпирическая ошибка получается куда меньше теоретической, можно выдвинуть гипотезу о том, что это не случайность и оптимальное квантование действительно всегда дает меньшую ошибку. (для большей уверенности в этой гипотезе стоит посмотреть на ошибки, полученные равномерным квантованием для этого конкретного сигнала). Второй график может свидетельствовать о наличии более выстрого убывания ошибки.

9. Сравнение ошибок равномерного квантования и оптимального квантования Ллойда-Макса

Для этого применим равномерное квантование к "нормально-распределенному" сигналу и посчитаем ошибки.

```
quantorization_signal_un = zeros(4,length(original_signal));
for k = 1:1:4
    figure(18+k)
    quantorization_signal_un(k,:) = UniformQuantization(normal_signal, count_set, k, x_min,
x_max);
end
figure(23)
[error_empiric_un, error_theoretic_un] = QuantizationError(normal_signal,
quantorization_signal_un, x_min, x_max)

figure(24)
error_SNR_un = QuantizationErrorSNR(normal_signal, quantorization_signal_un, x_min, x_max)
```

Имеем

```
error_empiric_norm    error_empiric_un
    1.0386              13.2747
    0.3267              2.2324
    0.1118              0.5218
    0.0281              0.1213

error_theoretic_norm  error_theoretic_un
    8.3333              8.3333
    2.0833              2.083
    0.5208              0.5208
    0.1302              0.1302

error_SNR_norm        error_SNR_un
    5.5353            -5.5302
    10.5582           2.2122
    15.2150           8.5252
    21.2175           14.8613
```

Вывод:

Гипотеза подтверждается, оптимальное квантование действительно дает куда меньшую величину ошибки, нежели равномерное.