МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра механики, мехатроники и робототехники

УТВЕРЖДАЮ
проректор по учебной работе
_____ О.Г. Локтионова
«___» _____ 2016 г.

ИНФОРМАЦИОННЫЕ УСТРОЙСТВА И СИСТЕМЫ В МЕХАТРОНИКЕ И РОБОТОТЕХНИКЕ

Методические указания к выполнению лабораторных работ по дисциплине «Информационные устройства и системы в мехатронике и робототехнике» для студентов направлений 15.03.06

Курск 2016

УДК 621

Составители: С.И. Савин

Рецензент
Кандидат технических наук, доцент *Е.Н. Политов*

Информационные устройства и системы в мехатронике и робототехнике: методические указания к выполнению лабораторных работ по дисциплине «Информационные устройства и системы в мехатронике и робототехнике» / Юго-Зап. гос. ун-т; сост. С.И. Савин. Курск, 2016. 39 с.: ил. 21, табл. 2. Библиогр.: с.37.

Методические указания содержат сведения методах моделировании инкрементальных оптических энкодеров и алгоритмах обработки их сигналов.

Методические указания соответствуют требованиям программы, утверждённой учебно-методическим объединением (УМО).

Предназначены для студентов направлений подготовки 15.03.06 – Мехатроника и робототехника всех форм обучения.

Текст печатается в авторской редакции

Подписано в печать . Формат 60х84 1/16
Усл.печ.л. ___. Уч.-изд.л. ___ Тираж 100 экз. Заказ.
Бесплатно.
Юго-Западный государственный университет.
305040 Курск, ул. 50 лет Октября, 94

ИЗУЧЕНИЕ АЛГОРИТМОВ ОБРАБОТКИ СИГНАЛА ИНКРЕМЕНТАЛЬНОГО ОПТИЧЕСКОГО ЭНКОДЕРА

Цель работы: *изучение алгоритмов обработки сигнала инкрементального оптического энкодера средствами математического пакета MATLAB.*

Объект исследования: *инкрементальный оптический энкодер.*

Аппаратные средства: *математический пакет MATLAB.*

1.1. Краткие теоретические сведения

Инкрементальные оптические энкодеры являются устройствами для измерения углового перемещения. В робототехнике их используют для измерения угла поворотов валов двигателей, относительного вращения отдельных звеньев робота и т.п. Особенностью инкрементальных энкодеров является то, что они не измеряют ориентацию твердого тела непосредственно, но позволяют определить насколько эта ориентация изменилась.

Принцип работы инкрементального оптического энкодера заключается в использовании диска с отверстиями, устанавливаемого вал, чье вращение необходимо измерить. Диск освещается с одно стороны, а с противоположной располагаются чувствительные элементы, определяющие наличие света. Датчик генерирует электрический сигнал равный опорному напряжению датчика или нулю, в зависимости о того, освещён ли чувствительный элемент. Один энкодер может иметь несколько чувствительных элементов, что позволяет определить направление вращения вала.

Устройство инкрементального оптического энкодера схематично изображено на рисунке 1.1.

Рисунок 1.1 устройство инкрементального оптического энкодера [1]

Математически работа такого энкодера описывается выражением:

$$\begin{cases} u = u_{\max} & \text{if} \quad \operatorname{mod}(\varphi, \Delta\varphi) < \alpha \cdot \Delta\varphi \\ u = 0 & \text{otherwise} \end{cases}, \qquad (1.1)$$

где u - напряжение на выходе датчика, $u_{\max}$ - опорное напряжение, φ - измеряемый угол, $\Delta\varphi$ - шаг измерения энкодера, α - константа, определяемая формой и расположением прорезей на диске оптического энкодера. Далее будем полагать, что $\alpha = 0.5$.

Величина $\Delta\varphi$ связана с числом отверстий на диске энкодера следующим образом:

$$\Delta\varphi = 2\pi / n, \qquad (1.2)$$

где n - число отверстий на диске энкодера.

1.2. Методика выполнения лабораторной работы

Реализуем рассмотренную выше модель энкодера в виде функции в среде MATLAB:

```
bit = 4;
resolution = 2^bit;
step = 2*pi / resolution;

    function u = Encoder(phi)
        remainder = mod(phi, step);
        if remainder < 0.5*step
            u = 0;
        else
            u = 1;
        end
    end
```

В представленном коде переменная `bit` означает разрядность датчика, переменная `resolution` – число отверстий на диске энкодера, а переменная `step` определяет величину $\Delta\varphi$.

Синтаксис `function u = Encoder(phi)` представляет собой заголовок функции, где `phi` это вход функции и `u` это выход функции.

Далее, промоделируем работу этого датчика:

```
dphi = 0.001;
Range = [0; 1*pi];
Count = floor((Range(2) - Range(1)) / dphi);

Res.U = zeros(Count, 1);
Res.phi = zeros(Count, 1);

for i = 1:Count
    phi = Range(1) + i*dphi;

    Res.phi(i) = phi;
    Res.U(i) = Encoder(phi);
end
```

Данный код трассирует значения угла в диапазоне, определяемом переменной `Range` с шагом `dphi`. В представленном коде переменная `Count` означает число итераций алгоритма, `Res` – структура, поля которой (`Res.U` и `Res.phi`) это массивы, в которых хранятся полученные в ходе моделирования данные. Переменная `phi` это фактический угол поворота вала. Строка `for i = 1:Count` означает объявление цикла.

Объединим написанную ранее функцию и показанный здесь код, добавим вывод графиков:

```
function IncrementalEncoder()
close all;

bit = 4;
resolution = 2^bit;
step = 2*pi / resolution;

    function u = Encoder(phi)
        remainder = mod(phi, step);
        if remainder < 0.5*step
            u = 0;
        else
```

```
            u = 1;
        end
    end

dphi = 0.001;
Range = [0; 1*pi];
Count = floor((Range(2) - Range(1)) / dphi);

Res.U = zeros(Count, 1);
Res.phi = zeros(Count, 1);

for i = 1:Count
    phi = Range(1) + i*dphi;

    Res.phi(i) = phi;
    Res.U(i) = Encoder(phi);
end

plot(Res.phi, Res.U, 'LineWidth', 1); grid on;
xlabel('phi');
ylabel('u');
end
```

Строка `plot(Res.phi, Res.U, 'LineWidth', 1)` представляет собой вызов функции `plot` строящей графики. Первый аргумент функции – координаты точек на графике по оси абсцисс, второй - координаты точек на графике по оси ординат. Параметр `'LineWidth'` и следующее за ним число определяют толщину линии графика.

Строки `xlabel('phi')` и `ylabel('u')` создают подписи на графике, строка `grid on` – сетку.

Строки `Res.U = zeros(Count, 1)` и `Res.phi = zeros(Count, 1)` реализуют аллокацию памяти для массивов. Функция `zeros` создает массив заданного размера заполненый нулями.

Результат работы этого кода показан на рисунке 2.

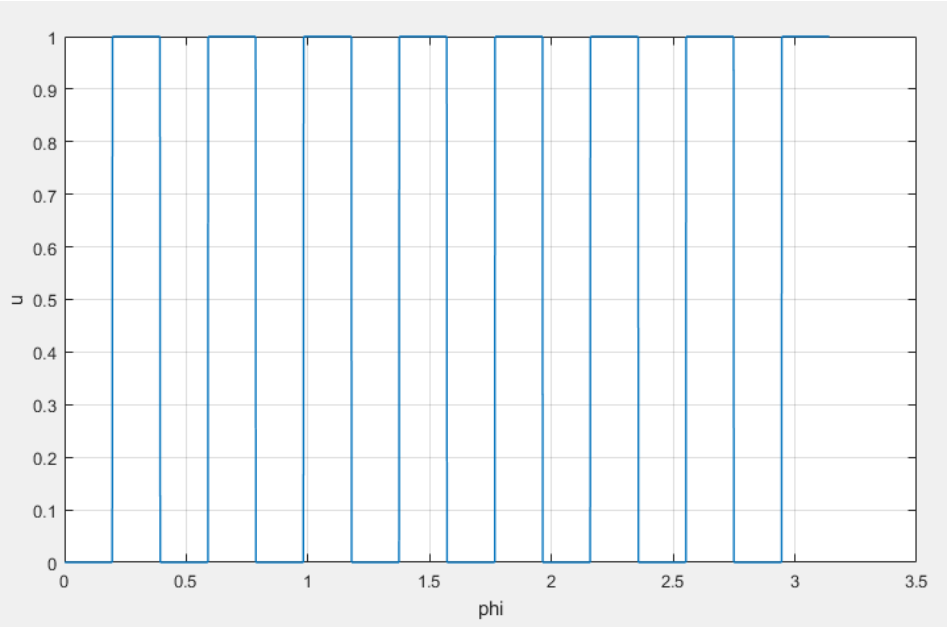

Рисунок 1.2 Зависимость выхода датчика от угла поворота вала

Заметим, что разрешение датчика выбрано крайне низким для того, чтобы улучшить наглядность графиков.

Дополним модель датчика, включив второй чувствительный элемент. Полученный код показан ниже.

```
function IncrementalEncoder2()
close all;

bit = 4;
resolution = 2^bit;
step = 2*pi / resolution;

    function u = Encoder(phi)
        remainder1 = mod(phi, step);
        remainder2 = mod(phi-0.1*step, step);
        u = zeros(2, 1);

        if remainder1 < 0.5*step
            u(1) = 0;
        else
            u(1) = 1;
        end
        if remainder2 < 0.5*step
            u(2) = 0;
        else
            u(2) = 1;
```

```
        end
    end

dphi = 0.001;
Range = [0; 1*pi];
Count = floor((Range(2) - Range(1)) / dphi);

Res.U = zeros(Count, 2);
Res.phi = zeros(Count, 1);

for i = 1:Count
    phi = Range(1) + i*dphi;
    u = Encoder(phi);

    Res.phi(i) = phi;
    Res.U(i, :) = u;
end

plot(Res.phi, Res.U, 'LineWidth', 1); grid on;
xlabel('phi');
ylabel('u');

end
```

Обратим внимание, что датчик теперь имеет два выходных сигнала, как показано на рисунке 3.

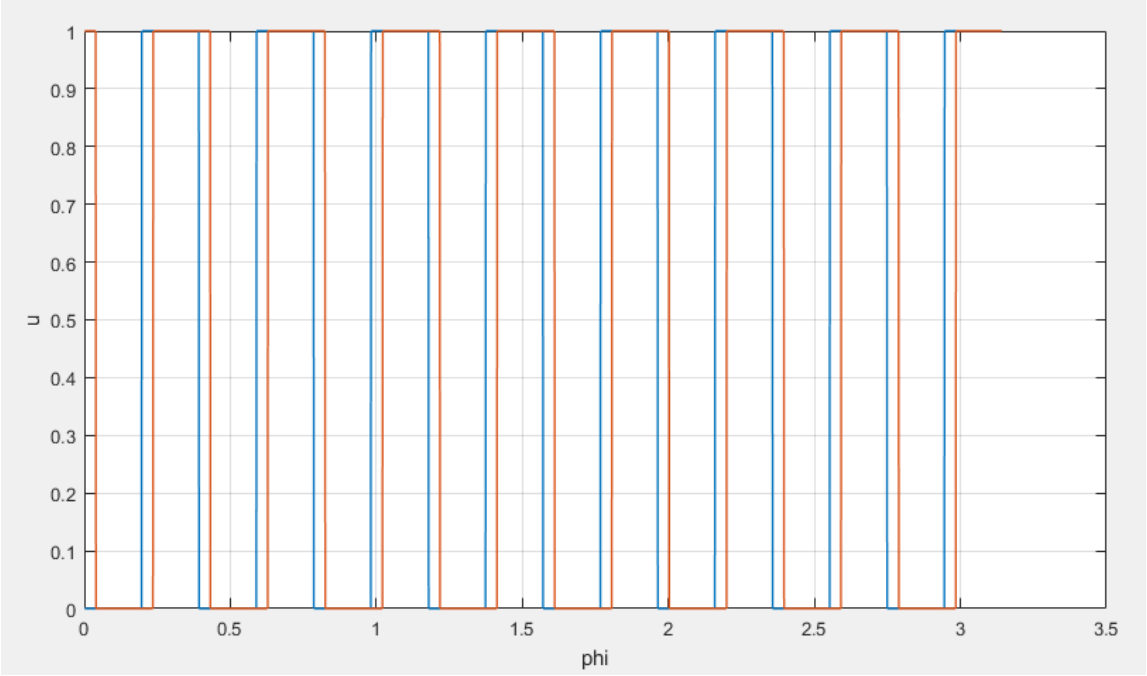

Рисунок 1.3 Зависимость выхода датчика от угла поворота вала

В показанных примерах мы рассматривали зависимость выхода датчика от угла поворота вала двигателя. Теперь, рассмотрим как выхода датчика зависит от времени при заданной угловой скорости вращение вала.

```
function IncrementalEncoder3()
close all;

bit = 4;
resolution = 2^bit;
step = 2*pi / resolution;

    function u = Encoder(phi)
        remainder1 = mod(phi, step);
        remainder2 = mod(phi-0.1*step, step);
        u = zeros(2, 1);

        if remainder1 < 0.5*step
            u(1) = 0;
        else
            u(1) = 1;
        end
        if remainder2 < 0.5*step
            u(2) = 0;
        else
            u(2) = 1;
        end
    end

dt = 0.001;
TimeRange = [0; 2];
Count = floor((TimeRange(2) - TimeRange(1)) / dt);

w = 1; phi0 = 0;

Res.U = zeros(Count, 2);
Res.phi = zeros(Count, 1);
Res.time = zeros(Count, 1);

for i = 1:Count
    t = TimeRange(1) + i*dt;
    phi = w*t + phi0;
    u = Encoder(phi);
```

```
    Res.phi(i) = phi;
    Res.U(i, :) = u;
    Res.time(i) = t;
end

plot(Res.time, Res.U, 'LineWidth', 1); grid on; hold
on;
plot(Res.time, Res.phi, 'LineWidth', 3); grid on;

xlabel('t');
ylabel('phi, u');

end
```

Переменная `w` является угловой скоростью вала, `phi0` – его фаза, `TimeRange` определяет временной интервал для которого производится моделирование.

Обратим внимание, что теперь угол поворота вала вычисляется как функция времени: `phi = w*t + phi0`. Также, время сохраняется в отдельный массив `Res.time`. Здесь мы строим 3 различных графика. Первые два – выход с датчика, строятся командой `plot(Res.time, Res.U, 'LineWidth', 1)`. Третий график строится командой `plot(Res.time, Res.phi, 'LineWidth', 3)`. Обратим внимание, что третий график построен линией толщиной 3, как это задано в коде. Также заметим, что для того, чтобы третий график не заменил первые два необходимо вызвать команду `hold on`.

На рисунке 1.4 показан результат работы приведённого кода.

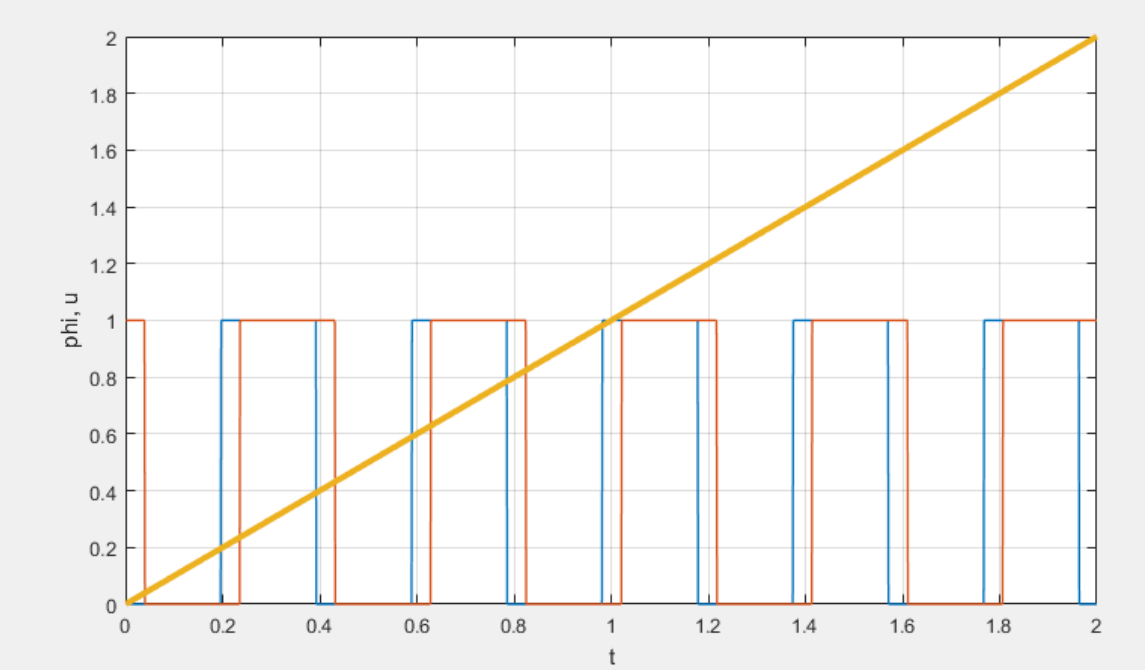

Рисунок 1.4 Зависимость выхода датчика и угла поворота вала от времени.

Реализуем простой алгоритм обработки сигнала с энкодера. Введем счетчик, который будет инкрементироваться каждый раз когда сигнал с одного из выходов датчика меняется. Код, реализующий этот алгоритм, приведён ниже.

```
function IncrementalEncoder4()
close all;

bit = 4;
resolution = 2^bit;
step = 2*pi / resolution;

    function u = Encoder(phi)
        remainder1 = mod(phi, step);
        remainder2 = mod(phi-0.1*step, step);
        u = zeros(2, 1);

        if remainder1 < 0.5*step
            u(1) = 0;
        else
            u(1) = 1;
        end
        if remainder2 < 0.5*step
            u(2) = 0;
        else
```

```
            u(2) = 1;
        end
    end


dt = 0.001;
TimeRange = [0; 2];
Count = floor((TimeRange(2) - TimeRange(1)) / dt);

w = 1; phi0 = 0;

Res.U = zeros(Count, 2);
Res.phi = zeros(Count, 2);
Res.time = zeros(Count, 1);

index = 0; u0 = Encoder(phi0);

for i = 1:Count
    t = TimeRange(1) + i*dt;
    phi = w*t + phi0;
    u = Encoder(phi);

    if u(1) ~= u0(1)
        index = index + 1;
    end
    u0 = u;
    calculated_phi = index * step / 2;

    Res.phi(i, :) = [phi; calculated_phi];
    Res.U(i, :) = u;
    Res.time(i) = t;
end

plot(Res.time, Res.U, 'LineWidth', 1); grid on; hold
on;
plot(Res.time, Res.phi, 'LineWidth', 3); grid on;

xlabel('t');
ylabel('phi, u');
end
```

Переменная `u0` хранит предыдущее значение входа с датчика (значение, найденное на прошлой итерации алгоритма), переменная `index` – упомянутый выше счетчик, переменная `calculated_phi`

хранит значение угла поворота, вычисленное предложенным методом.

На рисунке 5 показан результат работы приведённого кода.

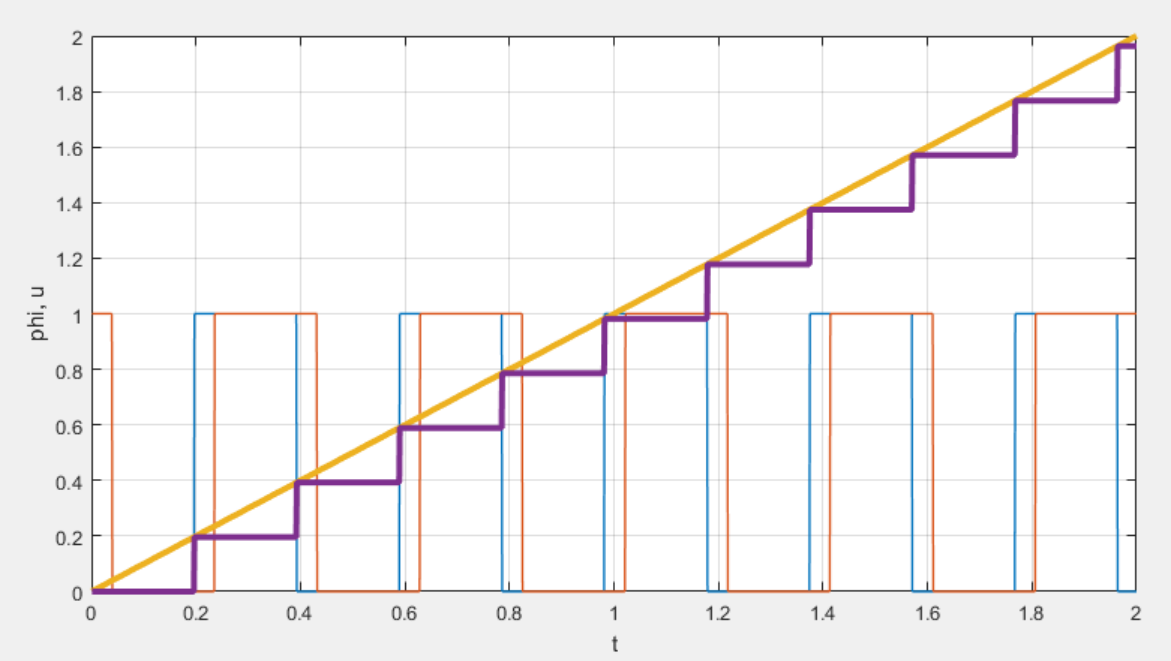

Рисунок 1.5 Зависимости выхода датчика и угла поворота вала (фактического и вычисленного) от времени.

Данный алгоритм работает корректно лишь при условии, что вал крутится в одном направлении. Используя оба канала датчика можно реализовать алгоритм, умеющий учитывать направление вращения вала двигателя. Соответствующий код приведён ниже.

```
function IncrementalEncoder5()
close all;

bit = 6;
resolution = 2^bit;
step = 2*pi / resolution;

    function u = Encoder(phi)
        remainder1 = mod(phi, step);
        remainder2 = mod(phi-0.1*step, step);
        u = zeros(2, 1);

        if remainder1 < 0.5*step
            u(1) = 0;
        else
            u(1) = 1;
        end
```

```
        if remainder2 < 0.5*step
            u(2) = 0;
        else
            u(2) = 1;
        end
    end


dt = 0.001;
TimeRange = [0; 2];
Count = floor((TimeRange(2) - TimeRange(1)) / dt);

w = -1; phi0 = 0;

Res.U = zeros(Count, 2);
Res.phi = zeros(Count, 2);
Res.time = zeros(Count, 1);

index = 0; u0 = Encoder(phi0); First = false;

for i = 1:Count
    t = TimeRange(1) + i*dt;
    phi = w*t + phi0;
    u = Encoder(phi);

    if (u(1) > u(2)) && ((u(1) > u0(1)) || (u(2) >
u0(2)))
        First = true;
    end

    if u(1) > u0(1)
        if First
            index = index + 1; First = false;
        else
            index = index - 1; First = false;
        end
    end

    u0 = u;
    calculated_phi = index * step;

    Res.phi(i, :) = [phi; calculated_phi];
    Res.U(i, :) = u;
    Res.time(i) = t;
end
```

```
plot(Res.time, Res.U, 'LineWidth', 1); grid on; hold
on;
plot(Res.time, Res.phi, 'LineWidth', 3); grid on;

xlabel('t');
ylabel('phi, u');
end
```

Основным отличием приведённого здесь кода является внедрение логической переменной (флага) `First`, определяющей направление вращения вала. Если `First` равна логической и истине `true` то первый из двух чувствительных элементов был освещен раньше чем второй, если же она имеет значение `false` то второй чувствительный элемент был освещен раньше чем первый. Это позволяет определить направление вращения вала. На рисунке 6 показан случай, когда вал вращается по часовой стрелке (угол поворота вала убывает).

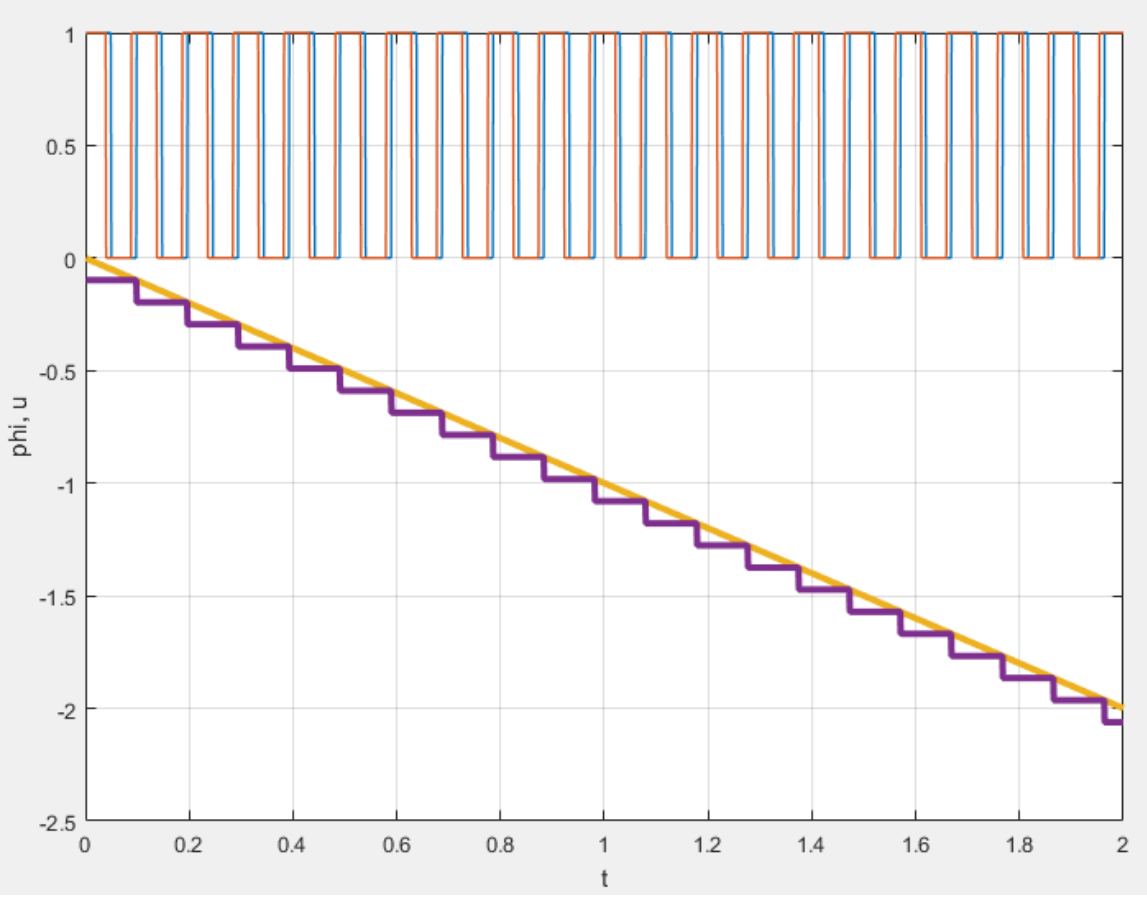

Рисунок 1.6 Зависимости выхода датчика и угла поворота вала (фактического и вычисленного) от времени.

На рисунках 7, 8 и 9 показано как алгоритм работает в случаях, когда движение вала описывается гармоническими законами.

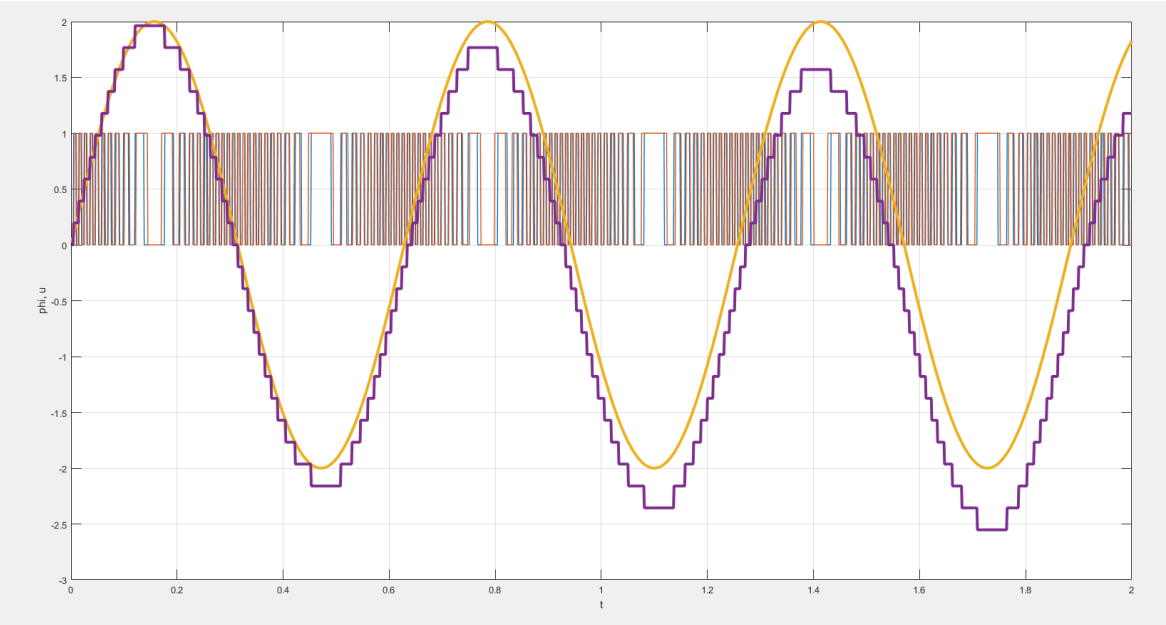

Рисунок 1.7 Зависимости выхода датчика и угла поворота вала (фактического и вычисленного) от времени, $\varphi = 2\sin(10t)$, n = 32

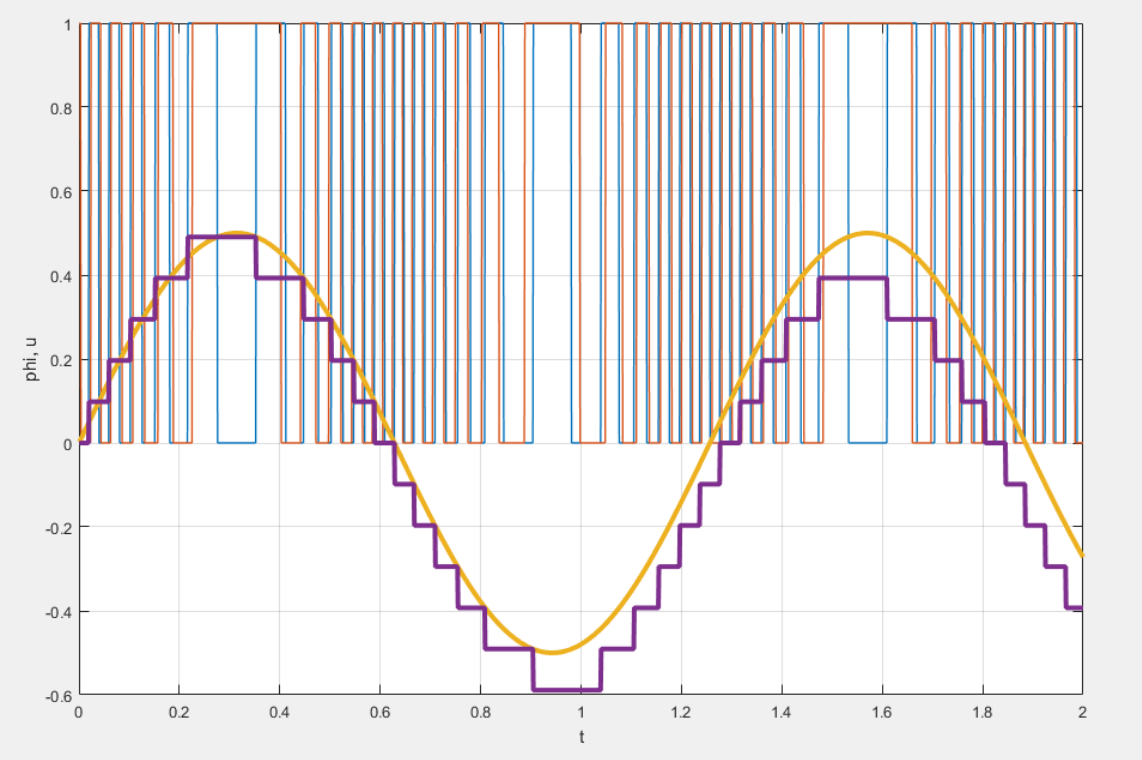

Рисунок 1.8 Зависимости выхода датчика и угла поворота вала (фактического и вычисленного) от времени, $\varphi = 0.5\sin(5t)$, n = 64

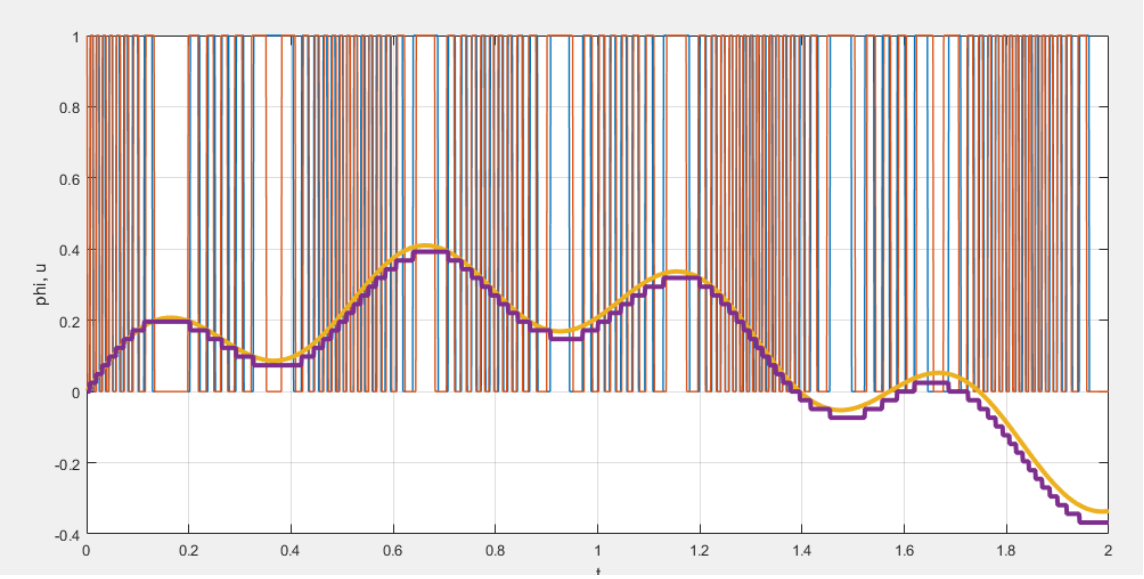

Рисунок 1.9 Зависимости выхода датчика и угла поворота вала (фактического и вычисленного) от времени,
$\varphi = 0.3\sin(2t) + 0.12\sin(12t)$, n = 256

Покажем, что рассматриваемый алгоритм требует, чтобы период колебаний угла поворота вала был существенно больше шага измерения датчика $\Delta\varphi$. На рисунке 10 приведён пример некорректной работы алгоритма из-за чрезмерно низкой точности датчика и чрезмерно высокой частоты колебаний угла поворота вала.

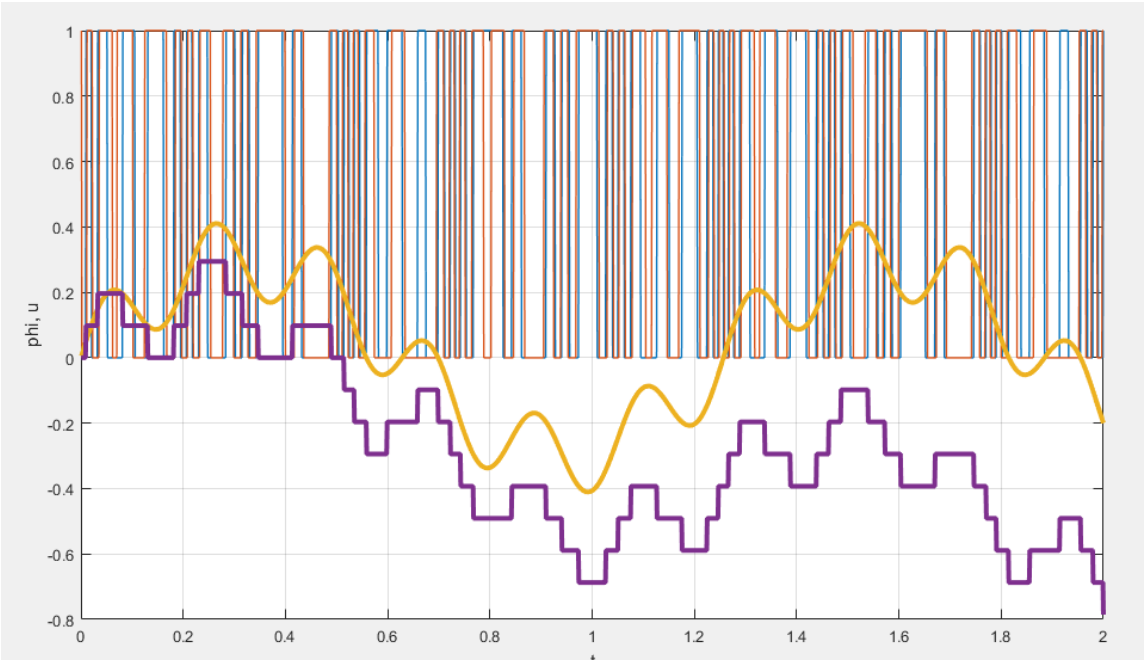

Рисунок 1.9 Зависимости выхода датчика и угла поворота вала (фактического и вычисленного) от времени,
$\varphi = 0.3\sin(5t) + 0.12\sin(30t)$, n = 64

1.3. Задание на выполнение лабораторной работы

Задание на выполнение лабораторной работы состоит в разложении в написании программ в среде математического моделирования MATLAB, реализующих:

- моделирование работы инкрементального оптического энкодера;
- моделирование работы алгоритма обработки данных с датчика.

Необходимо продемонстрировать при каких частотах происходит существенное накопление ошибки (разницы между вычисленным и фактическим углом поворота).

Функции, определяющие фактическую зависимость угла поворота вала следует выбирать из таблицы 1 в соответствии с номером студента в списке группы.

Таблица 1 Задания на выполнение лабораторной работы

№	Задание
1	$\varphi = 0.2\sin(2t) + 0.4\sin(5t)$
2	$\varphi = 0.6\sin(2.5t) + 0.2\sin(2t)$
3	$\varphi = 7\sin(6t) + 0.3\sin(10t)$
4	$\varphi = 2\sin(0.3t) + 7\sin(0.5t)$
5	$\varphi = 0.1\sin(1.3t) + 2\sin(0.3t)$
6	$\varphi = 0.3\sin(2.5t) + 0.2\sin(1.5t)$
7	$\varphi = 0.2\sin(3t) + 0.2\sin(1.7t)$
8	$\varphi = 0.35\sin(4.5t) + 0.25\sin(3.3t)$
9	$\varphi = 0.8\sin(7.1t) + 0.9\sin(0.45t)$
10	$\varphi = 0.95\sin(3t) + 0.5\sin(0.77t)$

ПРИМЕНЕНИЕ ФИЛЬТРА ГАУССА И ПРЕОБРАЗОВАНИЯ ФУРЬЕ ДЛЯ ВОССТАНОВЛЕНИЯ СИГНАЛА ОТ БЕЛОГО ШУМА

Цель работы: *изучение методов применения фильтра Гаусса для восстановления зашумленного сигнала.*

Аппаратные средства: *программный комплекс MATLAB.*

2.1. Краткие теоретические сведения

Современные системы автоматического управления во многих случаях полагаются на информацию, поступающую датчиков, для выработки управляющего сигнала. Точность информации, поступающей с датчика оказывает существенное влияние на качество работы системы автоматического управления. При этом на практике сигналы с датчиков зачастую оказываются «зашумлены». Под «зашумленным» сигналом подразумевают сигнал, представляющий собой смесь содержащего в себе полезную информацию сигнала и шума, сигнала не несущего полезной информационной нагрузки. Использование зашумленного сигнала в цепи обратной связи системы управления может вызвать ухудшение работы этой системы.

Для решения проблемы зашумленности сигнала рассмотрим метод фильтрации сигнала, использующий преобразование Фурье и фильтр Гаусса. Суть метода заключается в том, что над зашумленным сигналом производится преобразование Фурье для перехода к частотному домену, где на сигнал накладывается фильтр Гаусса. Подразумевается, что мы знаем какая область частот нас интересует. Над полученным сигналом производится обратное преобразование Фурье. В результате описанных действий получаем сигнал во временном домене, очищенный от шума.

Напомним, что преобразование Фурье описывается следующим образом:

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) \cdot e^{-i \cdot x \cdot \omega} dx \qquad (2.1)$$

Применение фильтра Гаусса равносильно умножению исходного сигнала на функцию Гаусса, описываемую следующим выражением:

$$\varpi(t) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{(t-t_0)^2}{2 \cdot \sigma^2}} \qquad (2.2)$$

Данный метод подходит для борьбы с белым шумом. При наличии других видов шумов следует проанализировать применимость и эффективность метода перед его применением.

2.2. Методика выполнения лабораторной работы

Работа выполняется в среде MATLAB. До начала работы с кодом рекомендуется выбрать рабочую папку и создать новый .m file в этой папке.

Сначала зададим функцию, имитирующую не зашумленный сигнал с датчика:

```
clc; clear; close all;
Time = 10;
dt = 0.001;

Count = floor(Time/dt);
Signal.Value = zeros(Count, 1);
Signal.Time = zeros(Count, 1);

for i=1:Count

    t = i*dt;
    y = 2 * sin(t);

    Signal.Value(i) = y;
    Signal.Time(i) = t;
end

plot(Signal.Time, Signal.Value, 'LineWidth', 3);
grid on;
```

Исполнение данного участка кода выведет на экран заданную функцию – в данном случае синусоиду.

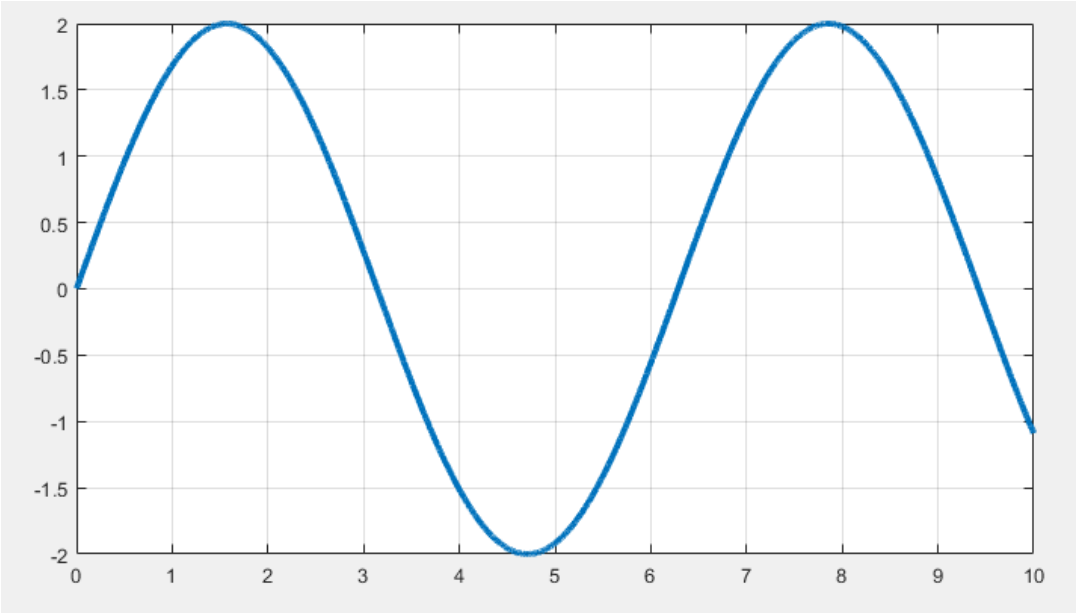

Рисунок 2.1 Исходный сигнал

Добавим к сигналу имитацию белого шума. Для этого прибавим к каждому элементу вектора A случайную величину в диапазоне от 0 до 2:

```
clc; clear; close all;
Time = 10;
dt = 0.001;

Count = floor(Time/dt);
Signal.Value = zeros(Count, 1);
Signal.ValueNoise = zeros(Count, 1);
Signal.Time = zeros(Count, 1);

for i=1:Count

    t = i*dt;
    y = 2 * sin(t);
    y_Noise = y + (rand-0.5)*2;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end

plot(Signal.Time, Signal.ValueNoise, 'LineWidth', 1);
hold on;
plot(Signal.Time, Signal.Value, 'LineWidth', 3);
```

```
grid on;
```

Исполнение данного участка кода выведет на экран отображение зашумленного сигнала:

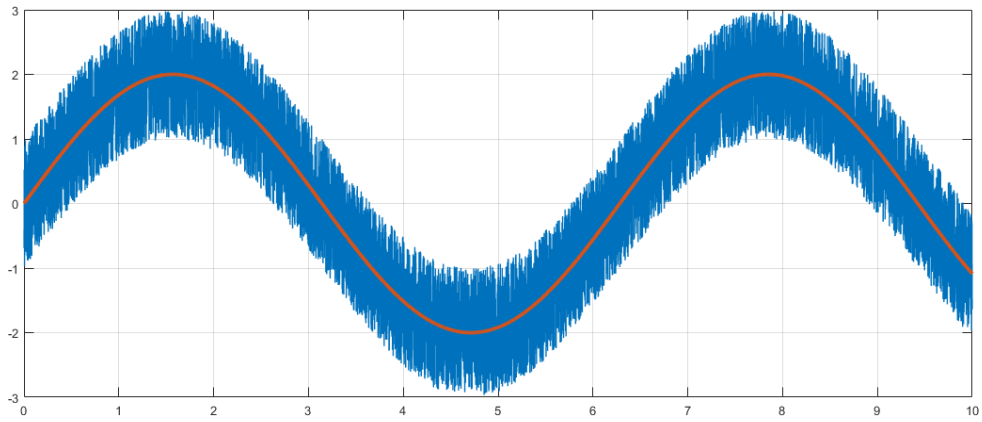

Рисунок 2.2 Зашумленный сигнал

Используем быстрое преобразование Фурье для перехода в частотный домен. Воспользуемся встроенными функциями пакета MATLAB – fft() и fftshift(). Последняя нужна для того, чтобы полученный спектр имел центр в нуле; функция fft() возвращает смещенный спектр, fftshift() позволяет это исправить.

```
clc; clear; close all;
Time = 10;
dt = 0.001;

Count = floor(Time/dt);
Signal.Value = zeros(Count, 1);
Signal.ValueNoise = zeros(Count, 1);
Signal.Time = zeros(Count, 1);

for i=1:Count

    t = i*dt;
    y = 2 * sin(t);
    y_Noise = y + (rand-0.5)*2;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
```

```
    Signal.Time(i) = t;
end


F = fft(Signal.ValueNoise);
F = fftshift(F);


subplot(3, 2, 1:2)
plot(Signal.Time, Signal.ValueNoise, 'LineWidth', 1);
hold on;
plot(Signal.Time, Signal.Value, 'LineWidth', 3);
title('Signal and noisy signal');
grid on;


subplot(3, 2, 3)
plot(abs(F));
title('Abs value of the spectrum');
grid on;
subplot(3, 2, 4)
plot(angle(F));
title('Phase value of the spectrum');
grid on;

subplot(3, 2, 5)
plot(imag(F));
title('imaginary value of the spectrum');
grid on;
subplot(3, 2, 6)
plot(real(F));
title('real value of the spectrum');
grid on;
```

Выполнение данного участка кода выведет на экран спектрограмму зашумленного сигнала:

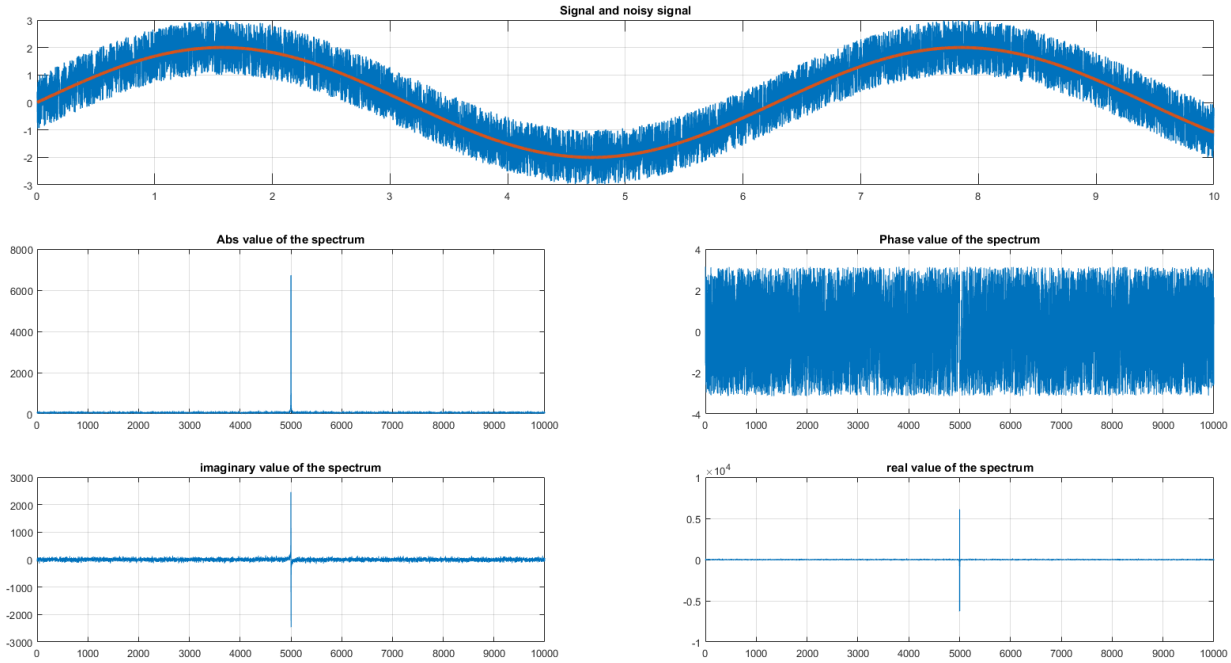

Рисунок 2.3 Зашумленный сигнал, его спектрограмма и её компоненты

Обратим внимание, что на спектрограмме присутствует один пик, соответствующий частоте полезного сигнала. Остальные частоты представлены равномерно, что характерно для белого шума.

Представленный ниже код генерирует фильтр Гаусса (Функцию Гаусса представленную в виде вектора той же размерности что и вектор, содержащий спектр зашумленного сигнала)

```
clc; clear; close all;
Time = 10;
dt = 0.001;

Count = floor(Time/dt);
Signal.Value = zeros(Count, 1);
Signal.ValueNoise = zeros(Count, 1);
Signal.Time = zeros(Count, 1);

for i=1:Count

    t = i*dt;
    y = 2 * sin(t);
```

```
    y_Noise = y + (rand-0.5)*2;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end


Scectrum = fft(Signal.ValueNoise);
Scectrum = fftshift(Scectrum);

GWP = 0.0001; %Это параметр определяет ширину Гауссовой
функции
Scectrum_size = length(Scectrum);

Filter = zeros(Scectrum_size, 1);
for i=1:Scectrum_size,
    Filter(i) = exp(-GWP*(i - (Scectrum_size / 2)).^2);
end



subplot(3, 2, 1:2)
plot(Signal.Time, Signal.ValueNoise, 'LineWidth', 1);
hold on;
plot(Signal.Time, Signal.Value, 'LineWidth', 3);
title('Signal and noisy signal');
grid on;


subplot(3, 2, 3)
plot(abs(Scectrum));
title('Abs value of the spectrum');
grid on;
subplot(3, 2, 4)
plot(angle(Scectrum));
title('Phase value of the spectrum');
grid on;

subplot(3, 2, 5)
plot(imag(Scectrum));
title('imaginary value of the spectrum');
grid on;
subplot(3, 2, 6)
```

```
plot(real(Scectrum));
title('real value of the spectrum');
grid on;


figure;
plot(Filter);
title('Filter');
grid on;
```

Выполнение данного участка кода выведет на экран Гауссову функцию:

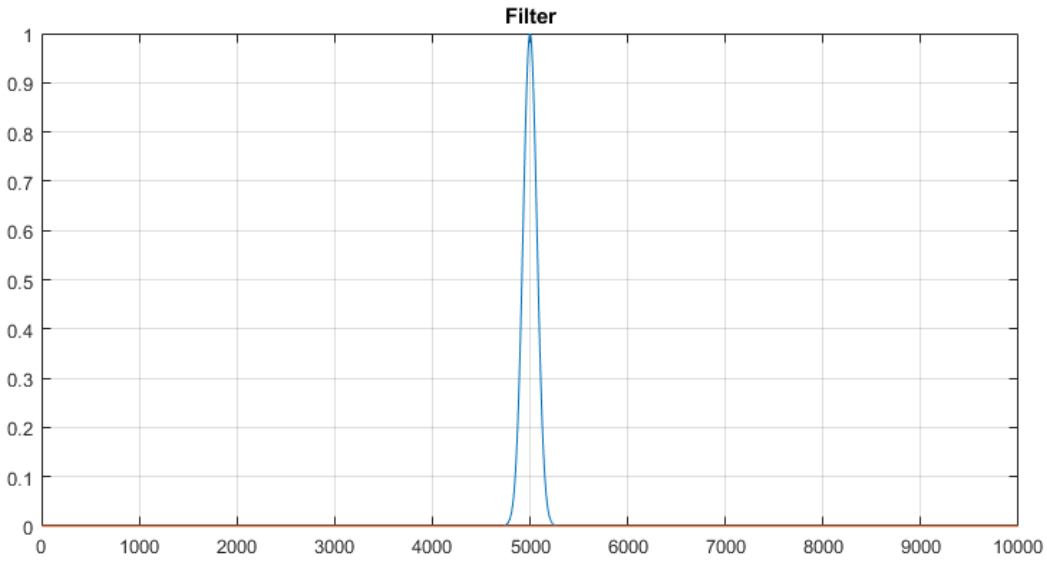

Рисунок 2.4 Функция Гаусса

Умножим спектр зашумленного сигнала на полученную Гауссову функцию и применим к результату обратное преобразование Фурье:

```
clc; clear; close all;
Time = 10;
dt = 0.001;

Count = floor(Time/dt);
Signal.Value = zeros(Count, 1);
Signal.ValueNoise = zeros(Count, 1);
Signal.Time = zeros(Count, 1);

for i=1:Count
```

```
    t = i*dt;
    y = 2 * sin(t);
    y_Noise = y + (rand-0.5)*2;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end


Scectrum = fft(Signal.ValueNoise);
Scectrum = fftshift(Scectrum);

GWP = 0.0001;
Scectrum_size = length(Scectrum);

Filter = zeros(Scectrum_size, 1);

for i=1:Scectrum_size,
    Filter(i) = exp(-GWP*(i - (Scectrum_size / 2)).^2);
end

Scectrum_Filtered = Scectrum.*Filter;
Scectrum_FilteredShifted =
ifftshift(Scectrum_Filtered);
Signal.ValueFiltered = ifft(Scectrum_FilteredShifted);


subplot(3, 2, 1:2)
plot(Signal.Time, Signal.ValueNoise, 'LineWidth', 1);
hold on;
plot(Signal.Time, Signal.Value, 'LineWidth', 3);
title('Signal and noisy signal');
grid on;


subplot(3, 2, 3)
plot(abs(Scectrum));
title('Abs value of the spectrum');
grid on;
subplot(3, 2, 4)
plot(abs(Scectrum_Filtered));
title('Abs value of the filtered spectrum');
```

```
grid on;

subplot(3, 2, 5:6)
plot(Signal.Time, Signal.Value, 'LineWidth', 3);
hold on;
plot(Signal.Time, Signal.ValueFiltered, 'LineWidth',
3);
title('Signal and filtered signal');
grid on;


figure;
plot(Filter);
title('Filter');
grid on;
```

Выполнение данной программы выведет на экран пять графиков – исходный сигнал, зашумленный сигнал, очищенный сигнал, Гауссову функцию, спектрограмму зашумленного и очищенного сигналов.

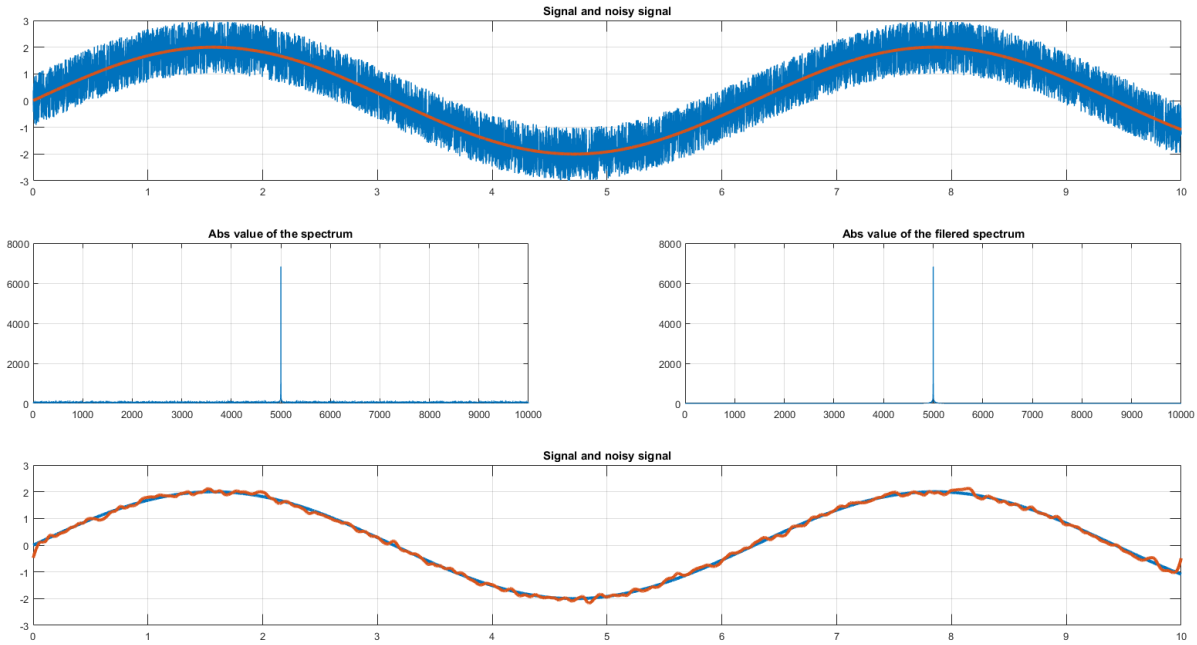

Рисунок 2.5 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

Увеличив амплитуду шума в 5 раз получим графики, показанные на рисунке ниже.

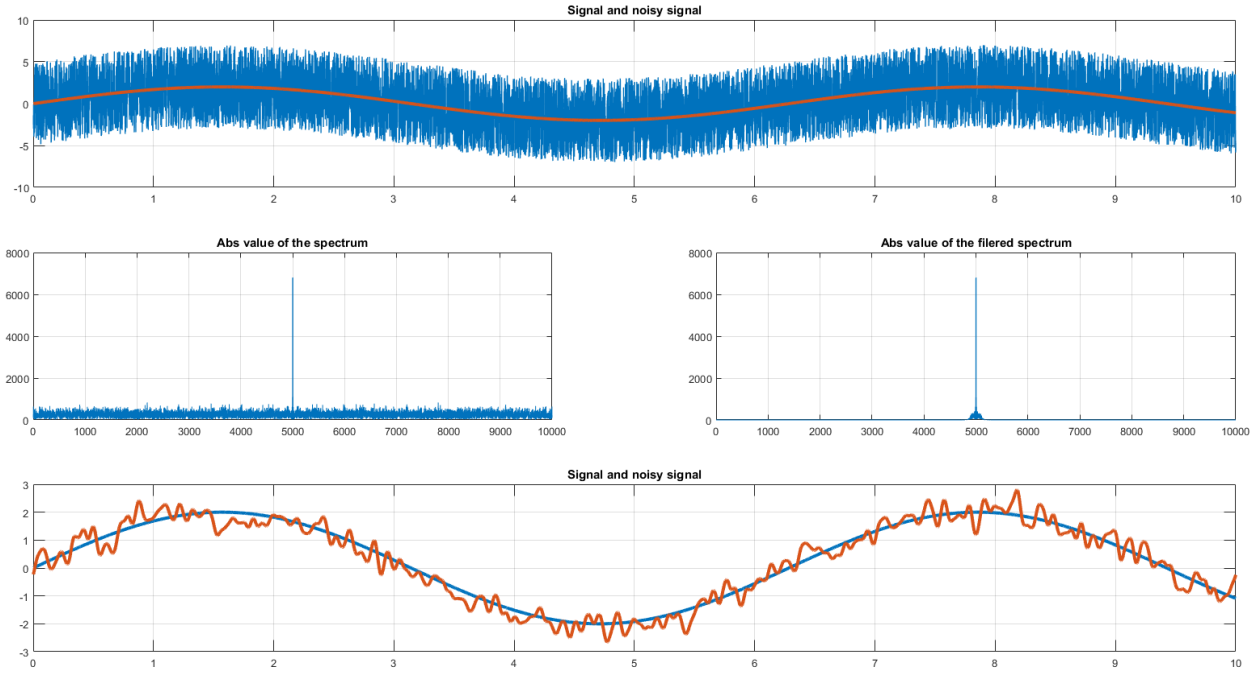

Рисунок 2.6 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

Добавим дополнительную гармонику в исходный сигнал и изменим ширину фильтра:

```
for i=1:Count

    t = i*dt;
    y = 2 * sin(t) + 1.2 * sin(8*t + pi/3);
    y_Noise = y + (rand-0.5)*10;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end

GWP = 0.00001;
```

Получим следующие графики:

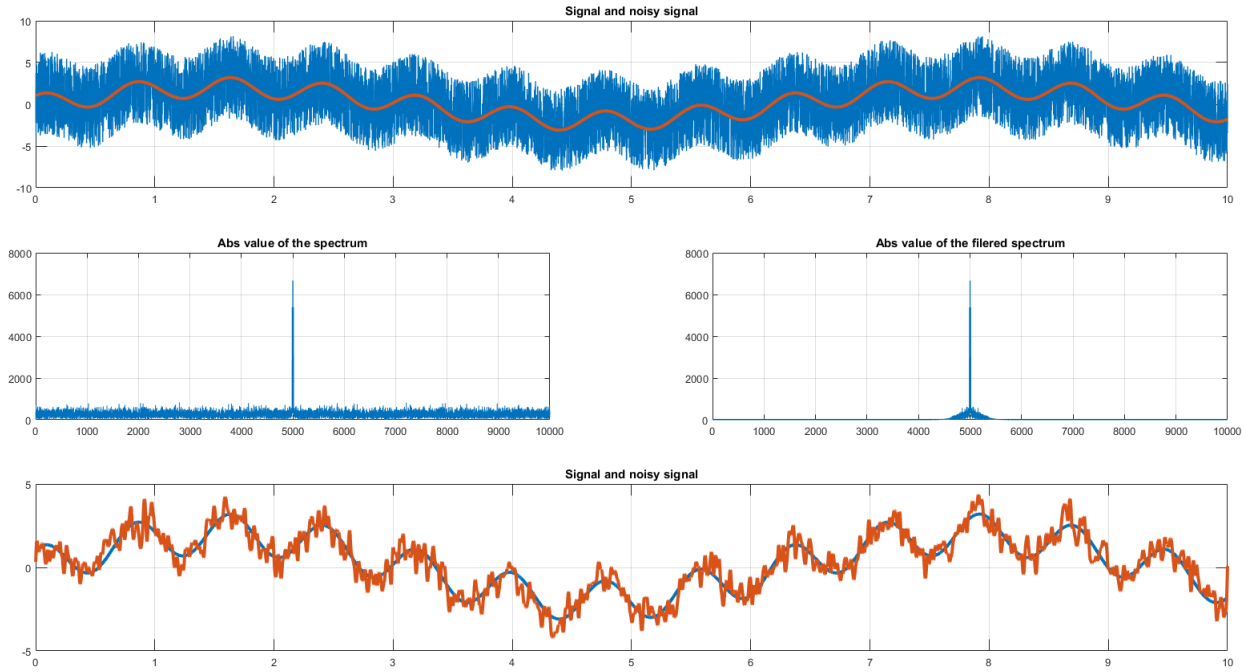

Рисунок 2.7 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

Как не сложно заметить, очищенный от шума сигнал не полностью совпадает с исходным сигналом. Это характерно для сигналов, полученных очисткой от белого шума с помощью наложения Гауссовой функции в частотном домене.

Уменьшим ширину фильтра (GWP = 0.001) и проведем вычисления повторно:

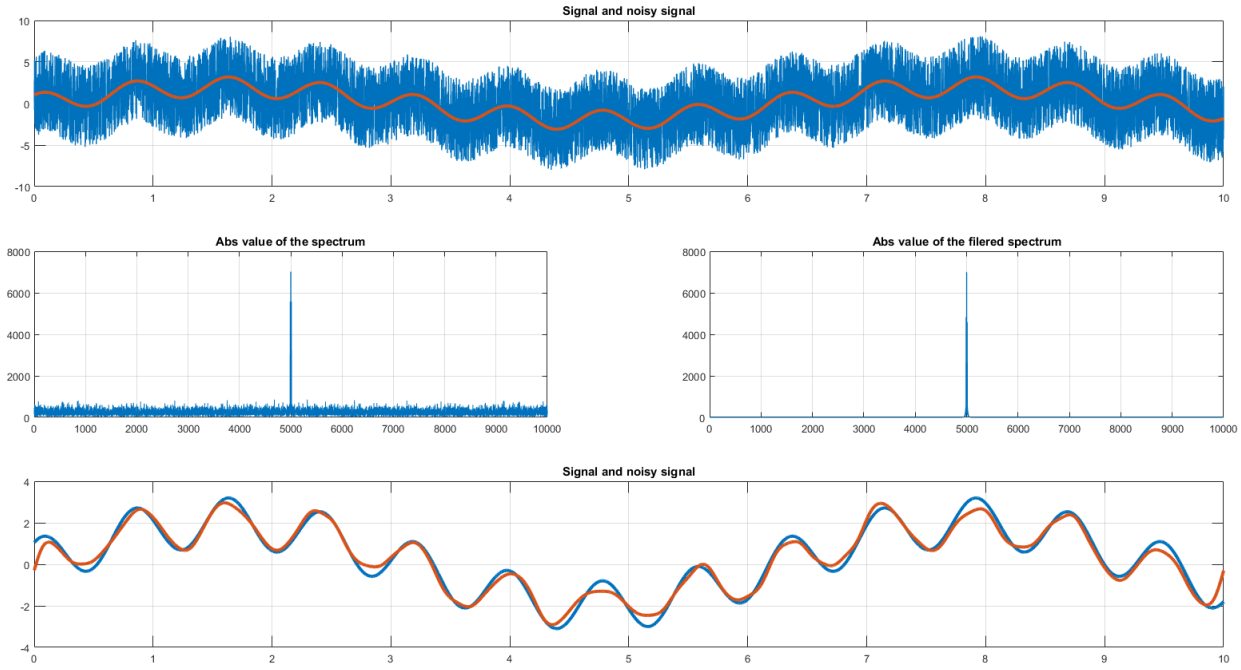

Рисунок 2.8 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

Заменим исходную функцию с синуса на меандр и проведем те же вычисления:

```
for i=1:Count

    t = i*dt;
    y = 2 * square(t);
    y_Noise = y + (rand-0.5)*10;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end
```

Получим следующие графики:

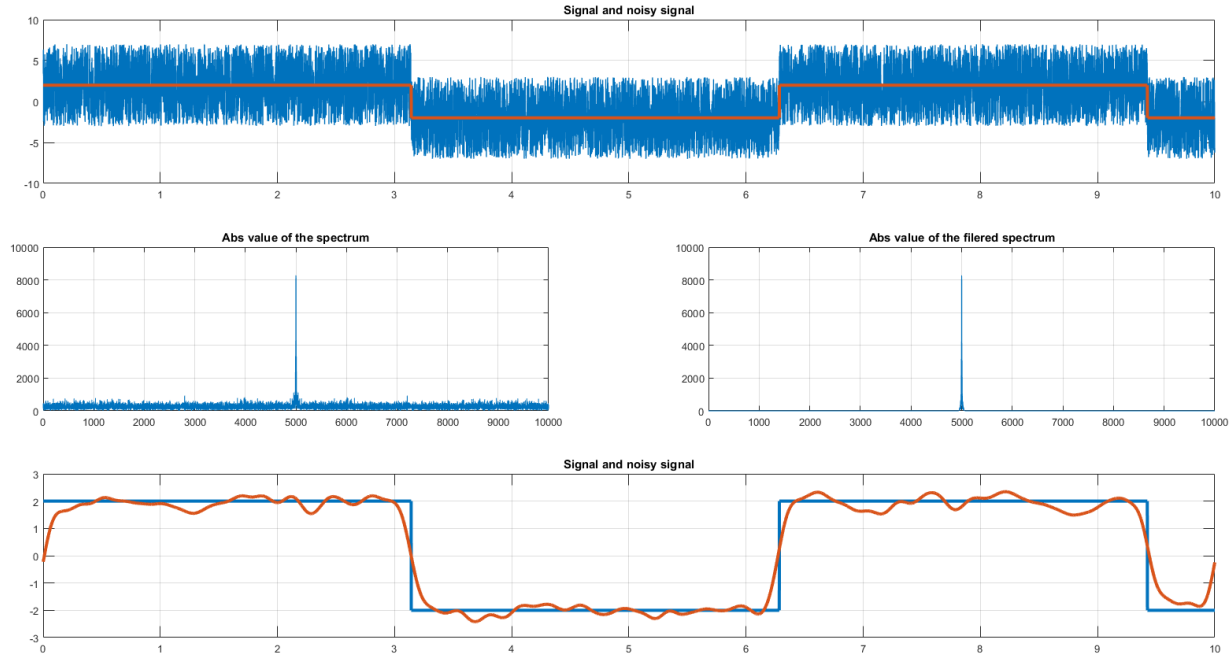

Рисунок 2.9 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

Добавим гармоническую составляющую и получим сигнал сложной формы.

```
for i=1:Count

    t = i*dt;
    y = 3 * square(t) + 1.2 * sin(8*t + pi/3);
    y_Noise = y + (rand-0.5)*10;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end
```

Для этого сигнала также произведем очистку спектра:

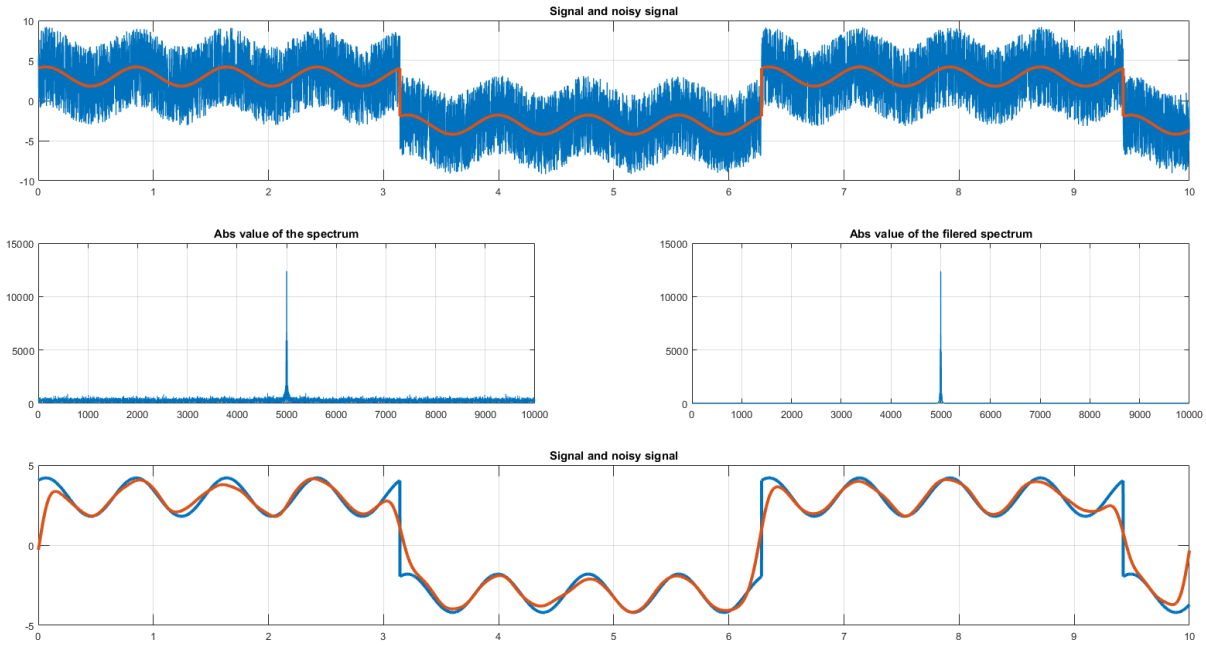

Рисунок 2.10 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

Заметим, что очищенный сигнал искажен. Рассмотрим случай, когда амплитуда белого шума меньше в 10 раз. Также, будем использовать значительно более «широкий» фильтр.

```
for i=1:Count

    t = i*dt;
    y = 3 * square(t) + 1.2 * sin(8*t + pi/3);
    y_Noise = y + (rand-0.5)*1;

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end

GWP = 0.00001;
```

Для этого случая возможно получить достаточно точное восстановление сигнала.

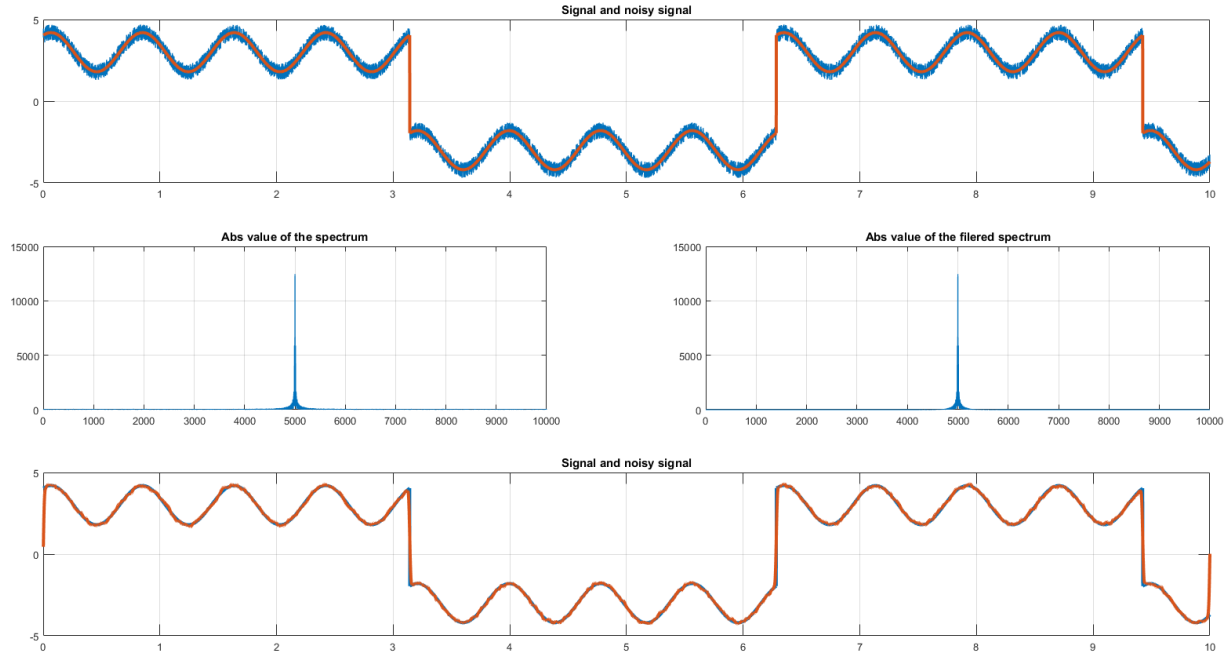

Рисунок 2.11 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

В данном случае форму исходного сигнала удалось сохранить со значительно большей точностью.
Наконец, рассмотрим случай, когда в роли шума выступает высокочастотный гармонический сигнал.

```
for i=1:Count

    t = i*dt;
    y = 3 * square(t) + 1.2 * sin(8*t + pi/3);
    y_Noise = y + 4*sin(1000*t);

    Signal.Value(i) = y;
    Signal.ValueNoise(i) = y_Noise;
    Signal.Time(i) = t;
end

GWP = 0.00001;
```

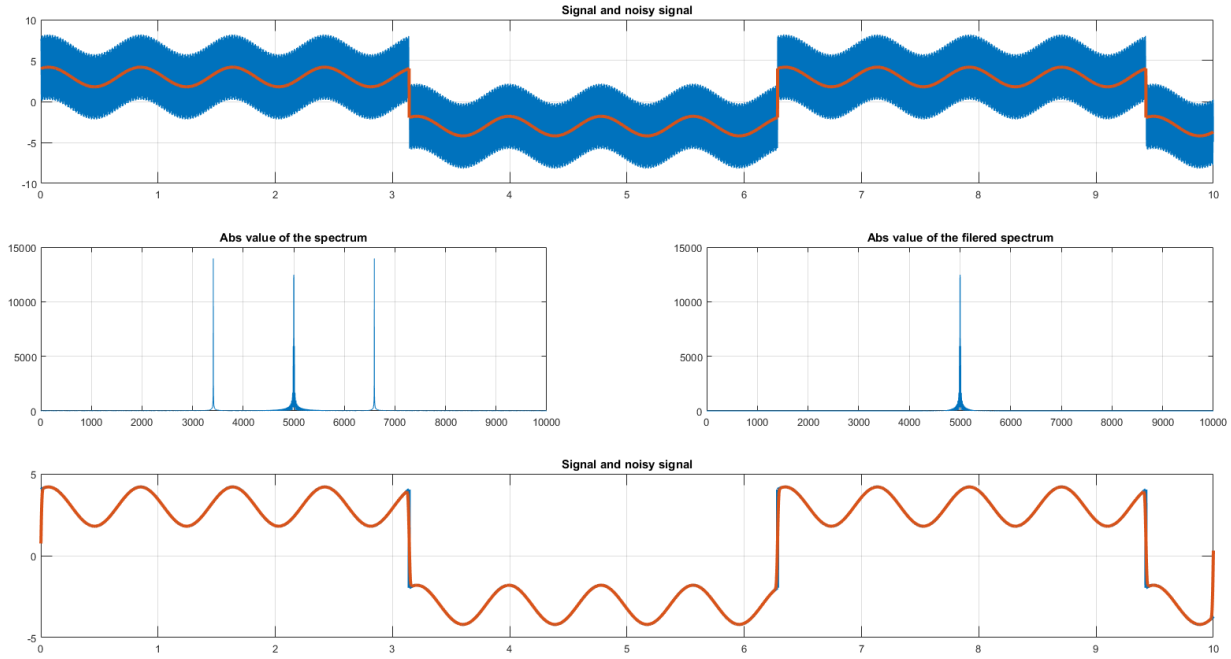

Рисунок 2.12 Слева на право и вниз: исходный сигнал и зашумленный сигнал, спектр зашумленного сигнала, спектр очищенного сигнала, очищенный сигнал.

Заметим, что, не смотря на значительную амплитуду шума, очистить спектр исходного сигнала оказалось возможным без существенного искажения формы исходного сигнала. Обратим внимание на характерные пики на спектрограмме зашумленного сигнала.

3. Задание на выполнение лабораторной работы

Задание на выполнение лабораторной работы состоит фильтрации гармонического сигнала от белого шума. Вид сигнала (функция $F(t)$) и амплитуду накладываемого белого шума η следует выбирать из таблицы 1 в соответствии с номером студента в списке группы.

Таблица 2.1 Задания на выполнение лабораторной работы

№	Задание
1	$F(t)=0.35\sin(10\cdot t)+0.15\sin(8\cdot t)+0.35\sin(6\cdot t),\ \eta=3$

2	$F(t)=0.15\sin(2\cdot t)+0.45\sin(8\cdot t)+0.85\sin(12\cdot t)+1.05\sin(0.5\cdot t),\ \eta=5$
3	$F(t)=0.25\sin(13\cdot t)+0.35\sin(12\cdot t)+0.75\sin(5\cdot t)+1.05\sin(4\cdot t),\ \eta=3$
4	$F(t)=0.45\sin(2\cdot t)+0.65\sin(0.5\cdot t)+0.05\sin(18\cdot t)+0.5\sin(9\cdot t),\ \eta=1.5$
5	$F(t)=0.85\sin(7\cdot t)+0.9\sin(4\cdot t)+0.1\sin(10\cdot t)+0.2\sin(12\cdot t),\ \eta=4.5$
6	$F(t)=0.6\cos(14\cdot t)+0.95\sin(2\cdot t)+0.5\sin(t)+0.7\sin(3\cdot t)+0.8\sin(12\cdot t),$ $\eta=5$
7	$F(t)=0.1\cos(4\cdot t)+0.7\sin(6\cdot t)+0.6\sin(t)+0.85\sin(12\cdot t)+0.9\sin(8\cdot t),$ $\eta=3$
8	$F(t)=0.5\cos(12\cdot t)+0.55\sin(17\cdot t)+0.65\sin(3\cdot t)+0.3\sin(15\cdot t)+0.45\sin(t),$ $\eta=4.5$
9	$F(t)=0.5\sin(2\cdot t)+0.3\sin(12\cdot t)+0.1\sin(0.5\cdot t),\ \eta=1.5$
10	$F(t)=0.2\sin(7\cdot t)+0.1\cos(6\cdot t)+0.3\cos(3\cdot t),\ \eta=3$

Оформление отчета о выполнении лабораторной работы

Требования к отчету:

- отчет содержит титульный лист, описание выполняемого задания, описание проделанной работы, анализ полученных результатов, выводы, список использованной литературы;
- отчет выполняется на листах формата А4, 14 кегль, одинарный межстрочный интервал;
- список литературы оформляется согласно ГОСТ 7.1-2003.

Рекомендуемая литература

1. http://www.kit-e.ru/articles/sensor/2005_9_34.php
2. https://www.mathworks.com/help/matlab/ref/plot.html
3. https://www.mathworks.com/help/matlab/ref/zeros.html
4. https://www.mathworks.com/help/matlab/ref/floor.html
5. https://www.mathworks.com/help/matlab/ref/mod.html
6. Воротников С.А. Информационные устройства робототехнических систем / Издательство МГТУ им. Н. Э. Баумана, 384 стр., 2005 г.
7. Яцун, С.Ф. Информационные устройства и системы в мехатронике [Текст]: учебное пособие / С.Ф. Яцун, П.А. Безмен // Курск: Юго-Зап. гос. ун-т. – 2013. – 240 с.
8. Яцун, С.Ф. Датчики и обработка сигналов в мехатронике [Текст]: учебное пособие / С.Ф. Яцун, П.А. Безмен // Курск: Юго-Зап. гос. ун-т. – 2014. – 238 с.
9. Шелухин, О.И., 2012. Моделирование информационных систем. Учебное пособие. Горячая линия-Телеком ББК: 32.882 УДК: 621.395.
10. Поршнев, С.В., 2011. MATLAB 7. Основы работы и программирования: учебник. М.: Бином. Лаборатория знаний, 2006.–320 с.–ISBN 5-9518-0137.
11. Кривилев, А.В., 2005. Основы компьютерной математики с использованием системы MATLAB. М.: Лекс-Книга.

12. Коробейников, А.Г., 2010. Разработка и анализ математических моделей с использованием MATLAB и MAPLE: Учебное пособие. СПб.: Санкт-Петербургский государственный университет информационных технологий, механики и оптики.
13. Коткин, Г.Л. and Черкасский, В.С., 2001. Компьютерное моделирование физических процессов с использованием MATLAB: Учеб. пособие/Новосиб. ун-т.
14. Савин С.И. Применение преобразования Фурье для обработки сигнала акселерометра: методические указания к выполнению практической и самостоятельной работы по дисциплине «Информационные устройства и системы в мехатронике» [Текст] / С.И. Савин// Юго-Зап. гос. ун-т; Курск, 2015. 14 с.
15. Савин С.И. Оконное преобразование Фурье; преобразование Габора: методические указания к выполнению практической и самостоятельной работы по дисциплине «Информационные устройства и системы в мехатронике» [Текст] / С.И. Савин// Юго-Зап. гос. ун-т; Курск, 2015. 12 с.
16. Савин С.И. Применение фильтра Гаусса для восстановления зашумленного сигнала: методические указания к выполнению практической и самостоятельной работы по дисциплине «Информационные устройства и системы в мехатронике» [Текст] / С.И. Савин// Юго-Зап. гос. ун-т; Курск, 2015. 10 с.
17. Савин С.И. Применение фильтра Гаусса для очистки изображения от белого шума: методические указания к выполнению практической и самостоятельной работы по дисциплине «Информационные устройства и системы в мехатронике» [Текст] / С.И. Савин// Юго-Зап. гос. ун-т; Курск, 2015. 16 с.
18. Савин С.И. Применение разложения в тригонометрический ряд Фурье для очистки сигнала от шума: методические указания к выполнению лабораторной работы по дисциплине «Информационные устройства и системы в мехатронике» [Текст] / С.И. Савин// Юго-Зап. гос. ун-т; Курск, 2015. 10 с.
19. Jatsun S. Mathematical model of two-links mechanism movement at discrete control actions Jatsun S., Savin S., Bezmen P. / Proceedings of the International Conference on Pure Mathematics

- Applied Mathematics (PM-AM 2015), Vienna, Austria, 2015, pp 146-149

20. Jatsun, S., Savin, S. and Yatsun, A., 2015, October. Study of a nonlinear control system for unbalanced two-link mechanism. In System Theory, Control and Computing (ICSTCC), 2015 19th International Conference on (pp. 180-185). IEEE.