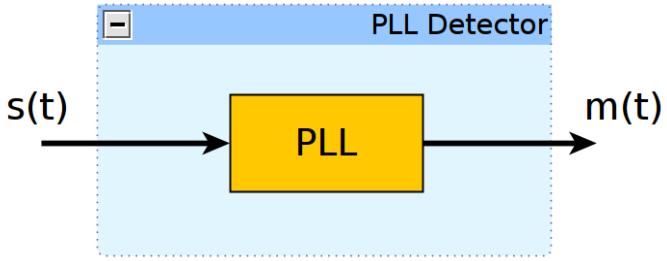
Демодуляция ЧМ-сигналов

Phase Locked Loop Detector

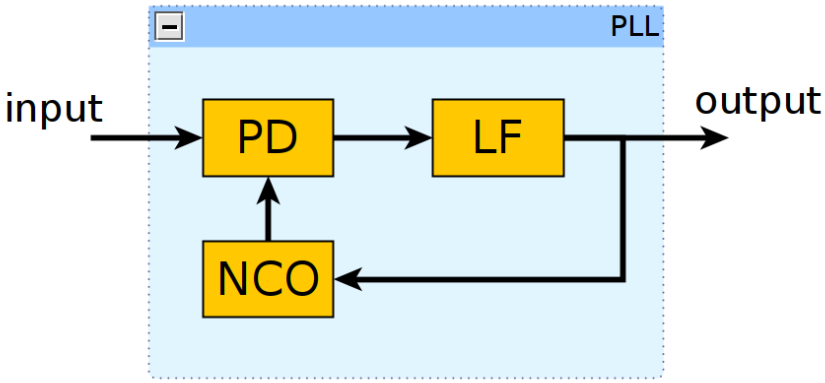
Для запуска скриптов необходимо корневую папку репозитория сделать рабочей папкой Matlab!

1. Phase Locked Loop (PLL) Detector

Наиболее качественный прием FM-сигналов с точки зрения помехоустойчивости обеспечивает демодулятор на основе фазовой автоподстройки частоты (phase locked loop - PLL). PLL отслеживает мгновенную частоту сигнал, которая при частотной модуляции задается информационным сообщением. Данный приемник также называют когерентным. Его схема представлена ниже:



Классическая PLL состоит из трех основных блоков: фазового детектора (phase detector - PD), петлевого фильтра (loop filter - LF) и генератора, управляемого напряжением (numericaly controled oscillator - NCO). Фазовый детектор вычисляет разность фаз между несущей и NCO. Схема PLL представлена ниже:



Фазовый детектор можно реализовать с помощью обычного умножителя. Будем считать, что несущую можно представить в виде:

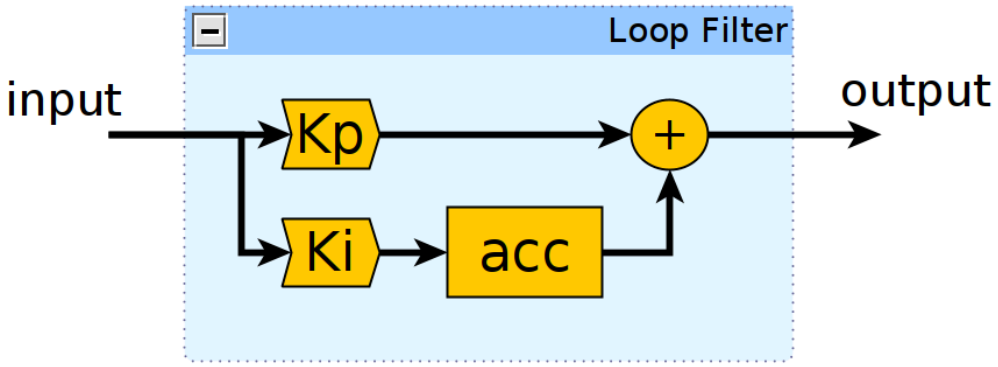
$$u_c = A \cdot \sin(\omega_c t + \phi_0) = A \cdot \sin(\Phi_c(t)).$$

Также пусть мгновенная фаза несущей в момент времени t_0 равна $\Phi_c(t_0)$ рад., а фаза NCO - $\Phi_{\text{NCO}}(t_0) = \Phi_c(t_0) - \Delta$ рад. Тогда получаем:

$$A \cdot \sin(\Phi_c(t_0)) \cdot \cos(\Phi_{\text{NCO}}(t_0)) = A \cdot \sin(\Phi_c(t_0)) \cdot \cos(\Phi_c(t_0) - \Delta) = A \cdot 0.5 \cdot \sin(\Delta) + A \cdot 0.5 \cdot \sin(2\Phi_c(t_0) - \Delta).$$

Слагаемое с удвоенной частотой можно убрать с помощью низкочастотного фильтра. Если этого не сделать, то оно в любом случае в дальнейшем будет удалено с помощью петлевого фильтра. Поэтому данным слагаемым можно пренебречь. То есть, отклик фазового детектора можно считать равным $A \cdot 0.5 \cdot \sin(\Delta)$. В случае, когда PLL находится в режиме синхронизации, расстройка по фазе будет близка к нулю, а значит $\sin(\Delta) \simeq \Delta$. Таким образом, сигнал на выходе детектора будет приближенно равен $A \cdot 0.5 \cdot \Delta$. Величину $k_d = A \cdot 0.5$ называют коэффициентом усиления детектора. Обратите внимание, что она зависит от амплитуды несущей.

Сигнал с выхода фазового детектора передается на вход петлевого фильтра, который удаляет шумы, присутствующий в сигнале, а также задает динамические характеристики PLL. Обычно фильтр реализуется в виде пропорциональной и интегрирующей ветвей, коэффициенты усиления которых равны k_p и k_i соответственно. Схема фильтра представлена ниже:



Эти коэффициенты рассчитываются на основе других характеристик PLL, а именно коэффициента демпфирования и шумовой полосы. Коэффициент демпфирования определяет степень осциляций при вхождении PLL в режим синхронизации. Шумовая полоса характеризует, как сильно будет ослаблятся шум на выходе PLL. Чем меньше полоса - тем меньше шума. Однако, уменьшение шумовой полосы приводит к существенному увеличению времени вхождения в режим синхронизации и уменьшает полосу захвата PLL. Формулы для расчета коэффициентов фильтра представлены ниже:

$$k_p = \frac{4 \cdot \zeta \cdot \mathrm{B}_n}{k_d \cdot \left(\zeta + \frac{1}{4 \cdot \zeta}\right)}, \quad k_i = \frac{4 \cdot \mathrm{B}_n{}^2}{k_d \cdot \left(\zeta + \frac{1}{4 \cdot \zeta}\right)^2},$$

где ζ - коэффициент демпфирования, $\mathrm{B}_n = \mathrm{B} \cdot f_s$ - нормированная шумовая полоса, B - шумовая полоса в герцах, f_s - частота дискретизации в герцах.

Сигнал с выхода петлевого фильтра изменяет частоту управляемого генератора и тем самым подстраивает его фазу для уменьшения ошибки Δ. Значание фазы NCO в аналитическом виде можно записать так:

$$\Phi_{\text{NCO}}(t) = \int_{-\infty}^{t} (\omega_0 + u_f(\tau))d\tau,$$

где ω_0 - частота NCO при отсутствии входнго воздействия, $u_f(t)$ - сигнал на выходе петлевого фильтра. Таким образом, управляемый генератор по сути представляет из себя интегратор. При синхронизации значение сигнала на входе управляемого генератора будет совпадать с мгновенной частотой принимаемого сигнала. То есть, значение $u_f(t)$ будет совпадать с информационным сообщением $m(t)$.

Рассмотрим как изменится структура фазовой автоподстройки частоты, если сигнал принимается квадратурным способом. Сигнал теперь комплексный с односторонним спектром. Отличае от случая действительнго сигнала заключается в устройстве фазового детектора. При квадратурном приеме несущую можно представить в виде комплексной экспоненты $A \cdot e^{j \cdot \Phi_c(t)}$. Выходной сигнал NCO также будет комплексным - $e^{-j \cdot \Phi_{\mathrm{NCO}}(t)}$. Как и ранее через $\Phi_c(t)$ и $\Phi_{\mathrm{NCO}}(t) = \Phi_c(t) - \Delta$ обозначены мгновенные фазы несущей и NCO. Фазовый детектор представляет из себя умножитель, после которого располается блок вычисления аргумента комплексного числа. Это можно записать в аналитическом виде:

$$s_{\mathrm{mix}} = A \cdot e^{j \cdot \Phi_c(t)} \cdot e^{-j \cdot \Phi_{\mathrm{NCO}}(t)} = A \cdot e^{j \cdot [\Phi_c(t) - \Phi_c(t) + \Delta]} = A \cdot e^{j \cdot \Delta},$$

$$s_d = \arg\{s_{\mathrm{mix}}\} = \arg\{A \cdot e^{j \cdot \Delta}\} = \Delta,$$

где s_{mix} - сигнал на выходе умножителя, а s_d - сигнал на выход фазового детектора.

Можно увидеть, что в случае квадратурного приема сигнал на выходе фазового детектора не зависит от амплитуды несущей, а коэффициент усиления детектора k_d равен единице. Остальная часть PLL полностью совпадает с реализацией для действительного принимаемого сигнала.

2. Демодуляция аудиосообщения

Рассмотрим демодуляцию аудиосообщения с помощью скрипта, реализующего PLL Detector.

В файлах Audio_FM_ModIdx_*.wav записаны частотно-модулированные сигналы с различными индексами модуляции. Частота несущей f_s равна 100 kHz, а частота дискретизации f_s = 441 kHz. Считаем, что прием выполняется квадратурным способ, то есть сигнал имеет вид:

$$s(t) = A_c \cdot \exp^{j \cdot \left(2\pi f_c t + 2\pi K_f \cdot \int_{-\infty}^{t} m(\tau) \cdot d\tau\right)}.$$

Чтобы PLL успевала следить за информационным сообщением $m(t)$, ее полоса пропускания должная быть больше полосы сообщения. Будем считать, что сообщение имеет полосу 20 kHz. Полосу PLL по уровню 3 dB возьмем с запасом, равной 40 kHz. Коэффициент демпфтирования пусть будет равен $1/\sqrt{2}$.

Рассчитаем шумовую полосу PLL. Для этого сначала найдем резонансную частоту контура PLL с помощью уравнения:

$$f_R = \frac{f_B}{\sqrt{1 + 2 \cdot \zeta^2 + \sqrt{(2 \cdot \zeta^2 + 1)^2 + 1}}},$$

где f_R - резонансная частотая контура в Hz, f_B - полоса пропускания контура по уровню 3 dB в Hz, ζ - коэффициент демпфирования.

Зная резонансную частоту, можем посчитать шумовую полосу PLL по слудующей формуле:

$$B = \frac{2\pi \cdot f_R \cdot \left(\zeta + \frac{1}{4\zeta}\right)}{2},$$

где B - шумовая полоса в Hz. В нашем примере шумовая полоса будет равна 64 kHz.

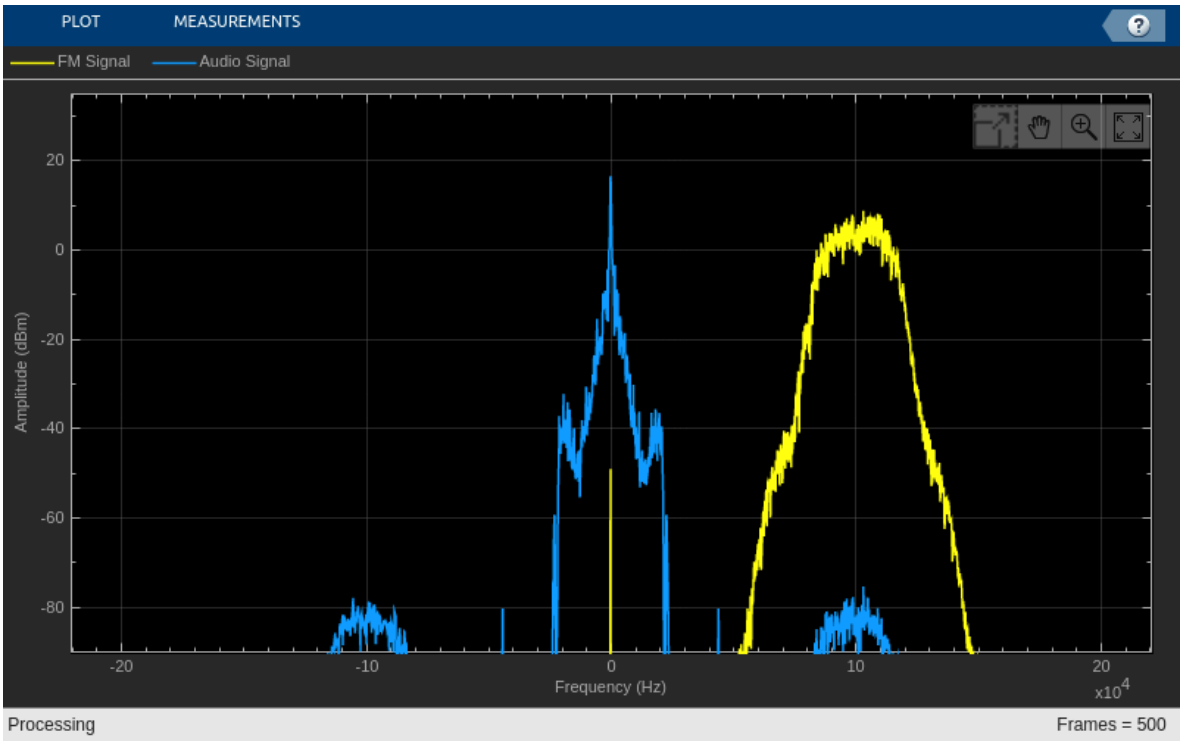
Для квадратурной фазовой автоподстройки частоты был создан Matlab System Object, описание которого находится в файле ComplexPLL.m. Данный System Object в качестве параметров принимает частоту дискретизации, коэффициент демпфирования, шумовую полосу и частоту NCO при отсутствии входного воздействия. На выходы System Object выдается сигналы на выходов петлевого фильтра и NCO.

Сигнал с выхода фильтра подается на дециматор. После децимации и удаления постоянной составлющей получаем информационное сообщение.

```
clc; clear; close all;
addpath('matlab/demodulation');

SignalFrameSize = 10000;          % количество отсчетов чм-сигнала, получаемых за один раз
FramesNumber = 500;               % число обрабатываемых пачек данных
RateRatio = 10;                   % коэффициент увеличения частоты дискретизации
AudioAmp = 1;                     % коэффициент усиления аудиосигнала
NcoAmp = 1 / 20e3;                % коэффициент усиления сигнала с выхода NCO
PllBandwidth3dB = 40e3;           % полоса пропускания PLL по уровню 3 dB
DampingFactor = 0.7;              % декремент затухания PLL
ModIndex = '1';    % индекс модуляции

% имя считываемого файла
SignalFileName = sprintf('wav/Audio_FM_ModIdx_%s.wav', ModIndex);

% объект для считываения отсчетов аудиофайла
AudioReader = dsp.AudioFileReader(...
    SignalFileName, ...
    'SamplesPerFrame',SignalFrameSize...
    );

% дополнительные расчеты
SignalFs = AudioReader.SampleRate;              % получаем частоту дискретизации модулированного сигнала
AudioFs = SignalFs / RateRatio;                 % частота дискретизации аудиосообщения

% расчет параметров PLL
ResonanceFrequency = PllBandwidth3dB / sqrt(1 + 2*DampingFactor.^2 + sqrt((2*DampingFactor.^2 + 1).^2 +
NoiseBandwidth = ResonanceFrequency * (2*pi) * (DampingFactor + 0.25/DampingFactor) / 2;

PLL = ComplexPLL( ...
    'SampleFrequency', SignalFs, ...
    'NoiseBandwidth', NoiseBandwidth, ...
    'Dampingfactor', DampingFactor, ...
    'CentralFrequency', 100e3 ...
```

```
    );

% дециматор
DownSampler = dsp.SampleRateConverter(...
    'Bandwidth', 30e3, ...
    'InputSampleRate', SignalFs, ...
    'OutputSampleRate', AudioFs ...
    );

% объект для вычисления спектра
SpecEstimator = dsp.SpectrumEstimator(...
    'PowerUnits','dBm',...
    'FrequencyRange','centered',...
    'SampleRate',SignalFs);

% объект для отрисовки графиков
Plotter = dsp.ArrayPlot(...
    'PlotType','Line', ...
    'XOffset', -SignalFs/2, ...
    'YLimits', [-90, 35], ...
    'XLabel', 'Frequency (Hz)', ...
    'YLabel', 'Amplitude (dBm)', ...
    'ChannelNames', {'FM Signal', 'Audio Signal'}, ...
    'SampleIncrement', SignalFs/SignalFrameSize ...
    );

Message = [];

% запуск симуляции
for i = 1:FramesNumber
    % считывание отсчетов ЧМ-сигнала и выделение синфазного канала
    FmSignal = AudioReader();
    FmSignal = FmSignal(:,1) + 1j*FmSignal(:,2);

    % удаление постоянной составляющей
    FmSignal = FmSignal - mean(FmSignal);

    % частотная демодуляция с помощью PLL
    [~, NcoOutput] = PLL(FmSignal);

    % нормирование выхода NCO
    NcoOutput = NcoOutput .* NcoAmp;

    % понижение частоты дискретизации
    AudioSignal = DownSampler(NcoOutput);

    % убираем постоянную составляющую
    AudioSignal = AudioSignal - mean(AudioSignal);

    % формирование итогового сллбщения
    Message = [Message; AudioAmp * AudioSignal];

    % вычисление спектров
    SpectrumData = SpecEstimator([FmSignal NcoOutput]);
```

```
    % вывод результатов на график
    Plotter(SpectrumData)

    % задержка в 0.1 секунды для лучшей визуализации
    pause(0.01)
end
```



На спектрограмме желтым показан спектр частотно-модулированного сигнала, а синим - спектр восстановленного сообщения.

```
% проигрывание полученного сообщения
sound(Message, AudioFs);
```

На слух можно оценить, что сообщение восстанавливается без искажений.

3. Демодуляция FM-радио в Matlab

Ниже представлен скрипт, позволяющий прослушивать FM-радио с помощью RTL-SDR. Настройка на нужную радиостанцию выполняется с помощью переменной Fc, которая задает частоту несущей. Входной сигнал проходит через фильтр нижних частот для выделения нужной радиостанции. Отфильтрованный сигнал поступает на PLL Detector. В заключение сигнал децимируется, у него удаляется постоянная составляющая и он подается на звуковую карту.

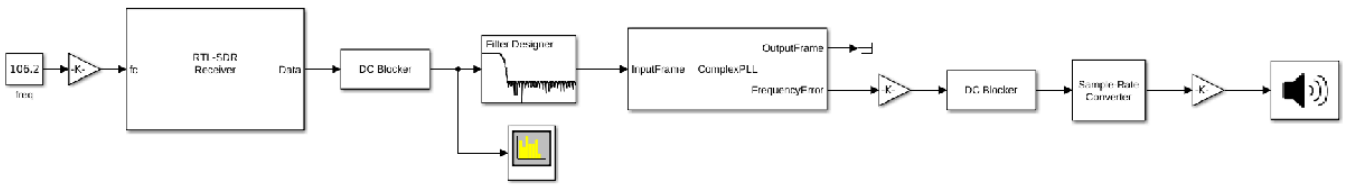
```
clc; clear; close all;
addpath('matlab/demodulation');

Fc = 106.2e6;                % частота несущей в Hz
SignalFs = 1.2e6;            % частота дискретизации RTL-SDR
AudioFs = 48e3;              % частота дискретизации демодулированного аудиосигнала
```

```
SignalFrameSize = 512*25;    % количество отсчетов чм-сигнала, получаемых за один раз
AudioAmp = 0.05;             % коэффициент усиления аудиосигнала
NcoAmp = 1 / 20e3;           % коэффициент усиления сигнала с выхода NCO
PllBandwidth3dB = 40e3;      % полоса пропускания PLL по уровню 3 dB
DampingFactor = 0.7;         % декремент затухания PLL

SDRRTL = comm.SDRRTLReceiver(...
    'RadioAddress', '0',...
    'CenterFrequency', Fc,...
    'EnableTunerAGC', true,...
    'SampleRate', SignalFs, ...
    'SamplesPerFrame', SignalFrameSize,...
    'OutputDataType', 'double' ...
);

% расчет коэффициентов и создание фильтра нижних частот
Fpass = 110e3;
Fstop = 160e3;
H = Audio_Lowpass_FIR_Coeff(SignalFs, Fpass, Fstop);
LowpassFIR = dsp.FIRFilter(H.Numerator);

% расчет параметров PLL
ResonanceFrequency = PllBandwidth3dB / sqrt(1 + 2*DampingFactor.^2 + sqrt((2*DampingFactor.^2 + 1).^2 +
NoiseBandwidth = ResonanceFrequency * (2*pi) * (DampingFactor + 0.25/DampingFactor) / 2;

PLL = ComplexPLL( ...
    'SampleFrequency', SignalFs, ...
    'NoiseBandwidth', NoiseBandwidth, ...
    'Dampingfactor', DampingFactor, ...
    'CentralFrequency', 0 ...
    );

% дециматор
DownSampler = dsp.SampleRateConverter(...
    'Bandwidth', 30e3, ...
    'StopbandAttenuation', 80, ...
    'InputSampleRate', SignalFs, ...
    'OutputSampleRate', AudioFs ...
    );

% объект для вычисления спектра
SpecEstimator = dsp.SpectrumEstimator(...
    'PowerUnits','dBm',...
    'FrequencyRange','centered',...
    'SampleRate',SignalFs);

% объект для отрисовки графиков
Plotter = dsp.ArrayPlot(...
    'PlotType','Line', ...
    'XOffset', -SignalFs/2, ...
    'YLimits', [-90, 35], ...
    'XLabel', 'Frequency (Hz)', ...
    'YLabel', 'Amplitude (dBm)', ...
    'ChannelNames', {'FM Signal'}, ...
```

```
    'SampleIncrement', SignalFs/SignalFrameSize ...
    );

% воспроизведение аудио сигнала
AudioSink = audioDeviceWriter(AudioFs);

% запуск симуляции
while(true)
    % получение отсчетов сигнала
    FmSignalData = SDRRTL();

    % вычисление спектров и вывод результатов на график
    SpectrumData = SpecEstimator(FmSignalData);
    Plotter(SpectrumData);

    % удаление постоянной составляющей
    FmSignalData = FmSignalData - mean(FmSignalData);

    % фильтрация
    FilteredData = LowpassFIR(FmSignalData);

     % частотная демодуляция с помощью PLL
    [~, NcoOutput] = PLL(FilteredData);

    % нормирование выхода NCO
    NcoOutput = NcoOutput .* NcoAmp;

    % уменьшение частоты дискретизации
    AudioData = DownSampler(NcoOutput);

    % удаляем постоянную составляющую
    AudioData = AudioData - mean(AudioData);

    % проигрывание данных
    AudioSink(AudioData * AudioAmp);

end
```

4. Демодуляция FM-радио в Simulink

В файле FM_SDR_Receiver_PLL_Detectoer.slx представлена Simulink-модель, позволяющая прослушивать FM-радио с помощью RTL-SDR. Все преобразования сигнала совпадают с теми, что ранее были описаны в Matlab скрипте.



Литература:

1. B. P. Lathi Modern Digital and Analog Communication Systems
2. R. Stewart, K. Barlee, D. Atkinson, L. Crockett Software Defined Radio using MATLAB® & Simulink and the RTL-SDR