

# Introduction to Baseband Processing

Phil Erickson, W1PJE<sup>1,2</sup>

Bill Liles, NQ6Z<sup>1,3</sup>

Ethan Miller, K8GU<sup>1</sup>

<sup>1</sup> HamSCI

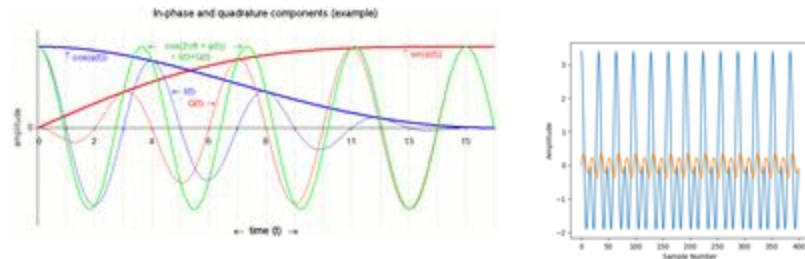
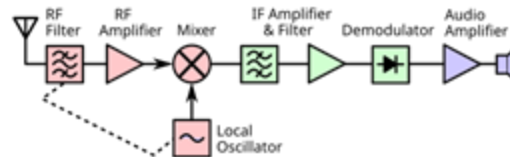
<sup>2</sup> MIT Haystack Observatory

<sup>3</sup> Liles Innovations, LLC

**HamSCI 2026 Workshop**

**Central Connecticut State University**

**2026-03-14**



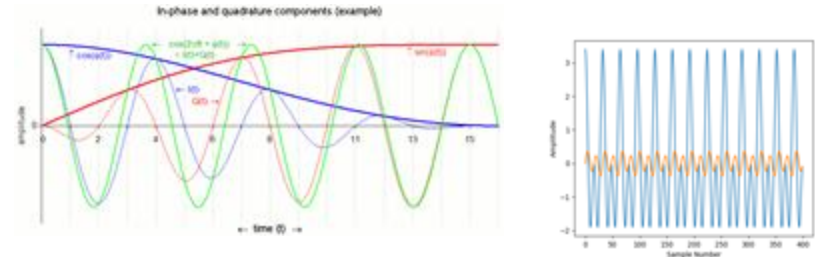
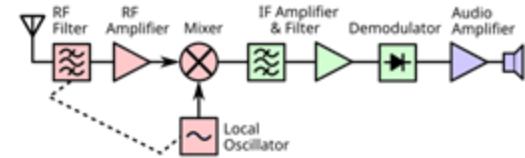
Python code:

```
h_lp = scipy.signal.firwin(num_taps, f_lp_cutoff,
fs=fs, pass_zero='lowpass')
# Complex sinusoid at the center frequency
complex_shift = np.exp(1j * 2 * np.pi * f_center
/ fs * n)
# Shift the filter taps to create the complex
bandpass filter
h_bp_complex = h_lp * complex_shift
```

# Introduction to Baseband Processing

## Outline

- Baseband Signals, Processing, Use
- IQ Signal Representations
- Time-Domain Filters
- AM Demodulation
- Demonstrations
- References and Resources



Python code:

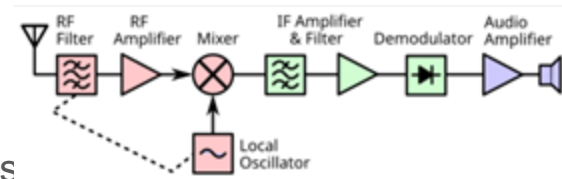
```
h_lp = scipy.signal.firwin(num_taps, f_lp_cutoff,
fs=fs, pass_zero='lowpass')
# Complex sinusoid at the center frequency
complex_shift = np.exp(1j * 2 * np.pi * f_center
/ fs * n)
# Shift the filter taps to create the complex
bandpass filter
h_bp_complex = h_lp * complex_shift
```

# Motivation for this Tutorial

- Much RF digital signal processing is done today not at original center frequency of transmission, but shifted to a lower frequency for ease of processing
- Analogous to classic radio receiver designs
  - E.g. superheterodyne (single mix) using IF frequencies = 455 kHz (AM), or 10.7 MHz (FM)
  - Allows processing to occur at a defined frequency ***no matter where*** the desired transmission is actually centered
- Most modern Software Defined Radios output in-phase/quadrature (I/Q) data streams
  - Includes ka9q-radio - used by HamSCI PSWS ecosystem
  - Both amplitude and phase information are contained in I/Q
- Understanding baseband complex signals in I/Q form and methods for their modification (filtering, frequency shifting, etc.) is useful



E. Howard Armstrong



Superheterodyne architecture  
(Wikipedia, CC0 1.0)

# Baseband Signal Definition

A **baseband** signal is a signal that has frequencies / information content located very near zero frequency (or DC).

## Original Definition:

- Signals with information coded at relatively low frequencies
- Usually from 0 Hz to some cutoff frequency

*Example: human voice (audio frequencies)*

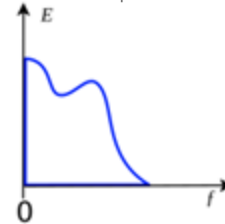
## Current / Working Definition for RF baseband:

- Signal occupies frequency usually from  $-f$  to  $+f$ , and centered at 0 Hz
- Most often: intermediate stage, before further processing
  - e.g. moving to passband (TX) or after RX before demodulation

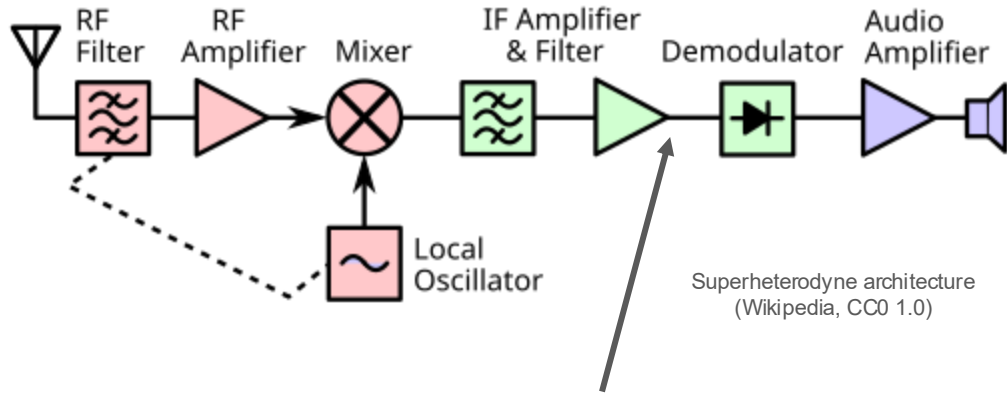
***This tutorial will deal with signals with both negative and positive frequencies around 0 Hz.***

Baseband signal  
(only positive frequencies shown  
here)

Wikipedia CC BY-SA 3.0



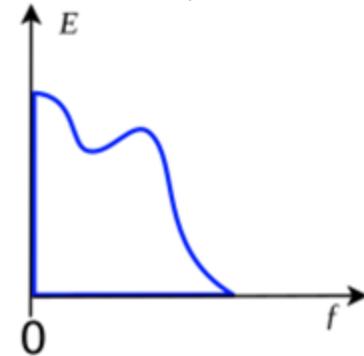
# Baseband Relation to the Superheterodyne



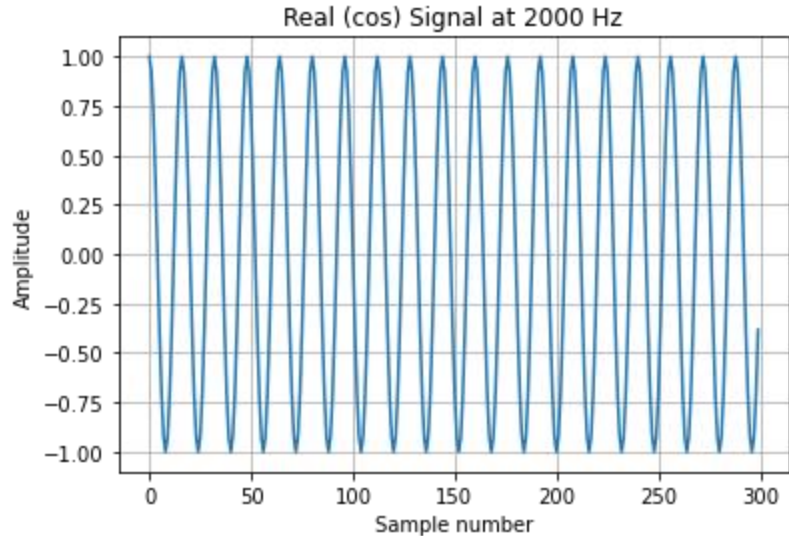
Superheterodyne receiver has an IF frequency usually centered at 455 kHz or 10.7 MHz.  
Baseband has an IF **centered at 0 Hz.**

Baseband signal  
(only positive frequencies shown here)

Wikipedia CC BY-SA 3.0

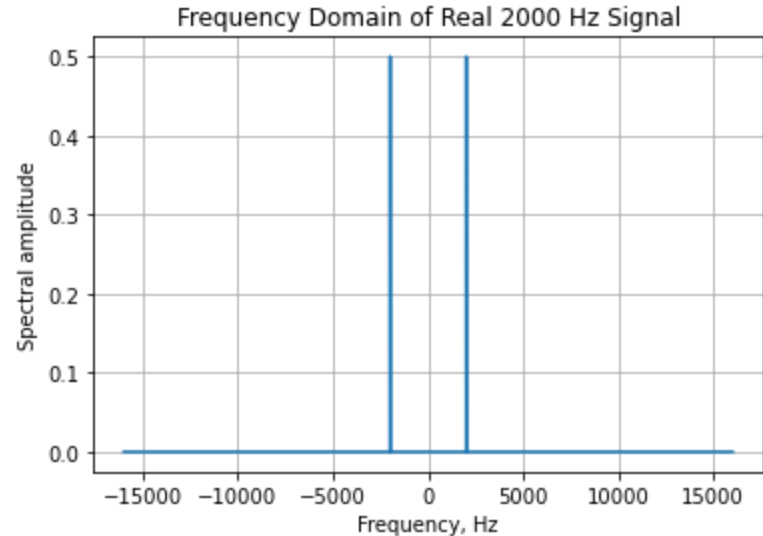


# Reminder: Frequency domain signal representation



e.g.  
FFT

←→

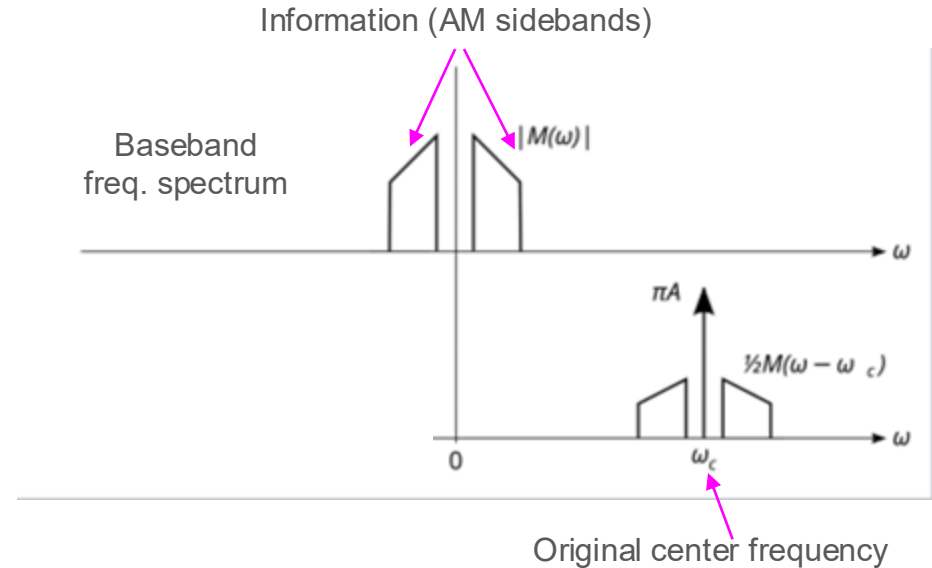


$$x(t) = \cos(2\pi f_c t)$$

$$X(f) \propto (\delta(f - f_c) + \delta(f + f_c))$$

# Baseband Processing Advantages

- Uses same practical advantage as superheterodyne
  - Easier to construct good filters and other signal processing methods
  - Processing occurs at a consistent (non-tuned) target frequency of 0 Hz
- Signals often have information content at a lower sampling rate compared to the RF sampling rate
  - Baseband processing allows use of a smaller final signal sampling rate than original RF sampling rate
  - Preserves information content



Amplitude modulated (AM) signal  
Wikipedia CC BY-SA 3.0

# Baseband signal processing use in current designs

- Direct conversion / homodyne / zero intermediate frequency (ZIF) receivers
  - Homodyne = mix the signal with a local oscillator at exactly the same center frequency as the incoming RF signal
  - “Zero beats” the incoming signal
- Many amateur/consumer radios provide IQ time series
  - Transceivers: QRP-Labs QMX(+), ...
  - SDR peripherals: kiwiSDR, RTL-SDR, HackRF
  - SDR software: GNURadio, SDRAngel, (digital\_rf)
- Most spectrum analyzers output IQ time series
  - All major manufacturers
  - e.g. R&S, Anritsu, Tektronix, Aaronia, SignalHound, Keysight



QRP-Labs QMX+

# In-phase and Quadrature (IQ) Signal Representation

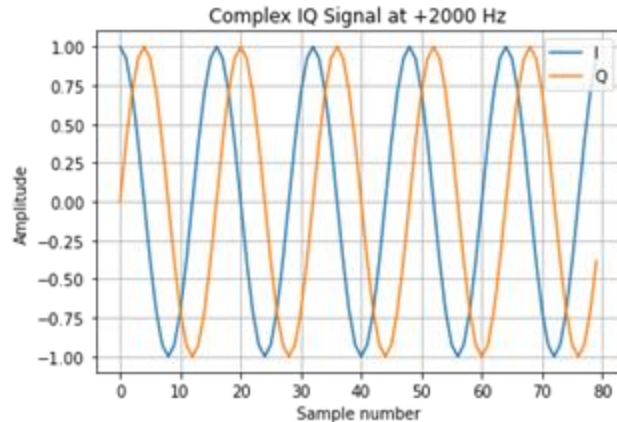
$$s(t) = A(t)\exp(j2\pi ft)$$



$$s(t) = I(t)\cos(2\pi ft) + jQ(t)\sin(2\pi ft)$$

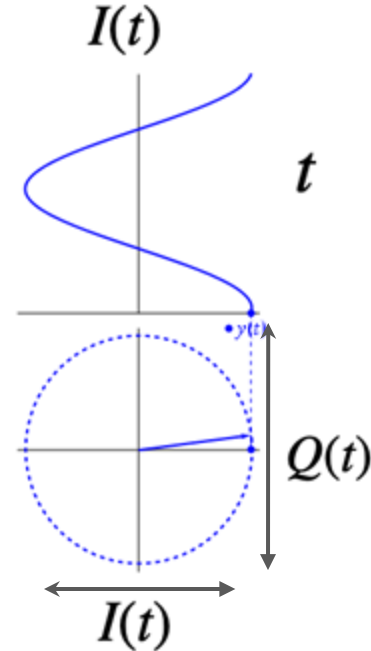
“I”

“Q”



Euler's formula

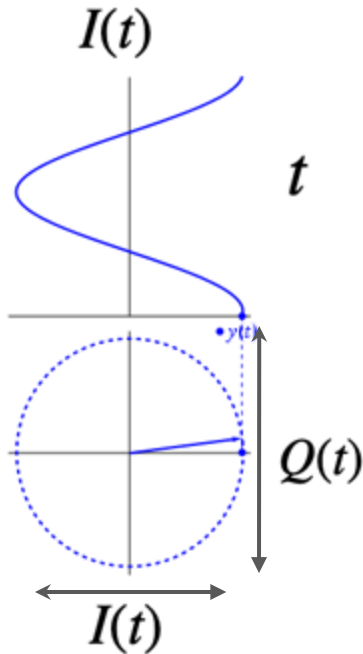
$$\exp(j2\pi ft) = \cos(2\pi ft) + j \sin(2\pi ft)$$



By Gonfer at English  
Wikipedia, CC BY-SA  
3.0,  
<https://commons.wikimedia.org/w/index.php?curid=11313700>

Conveys amplitude and phase (2 dimensions) of a complex domain signal

# Considerations for IQ Signal Representations



- Warning: with only one sampled signal (real domain), only 1 dimension of information
  - Inherent assumption is being made about incoming signal phase/frequency, such as frequency symmetry in complex domain
  - This assumption could be wrong
- Common example: AM vs SSB signals, when shift carrier freq. down to baseband / zero freq.
  - AM is symmetric around zero frequency
  - SSB is not (Upper vs lower sideband)
- Full  $I(t)$  and  $Q(t)$  representation allows relaxation of this implicit assumption - captures amplitude AND phase
- Also allows capture of signals with asymmetric frequency content

# Why does the spectrum of a real signal have positive and negative frequencies?

$$\exp(j2\pi f_c t) = \cos(2\pi f_c t) + j \sin(2\pi f_c t)$$

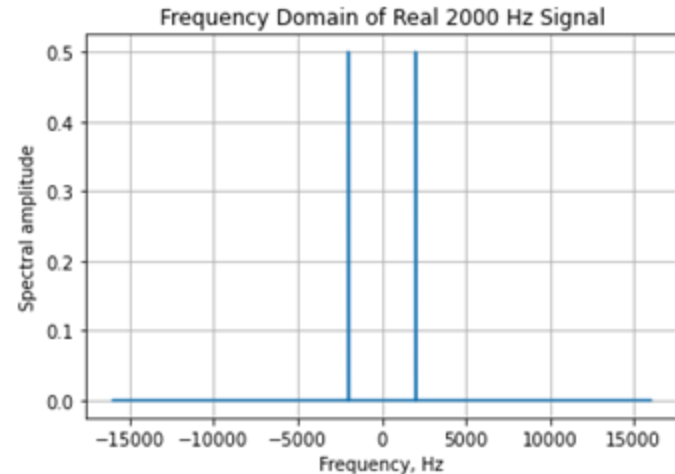
$$\exp(j2\pi(-f_c)t) = \cos(2\pi f_c t) - j \sin(2\pi f_c t)$$

$$\exp(j2\pi f_c t) + \exp(j2\pi(-f_c)t) = 2 \cos(2\pi f_c t)$$



$$\cos(2\pi f_c t) = [\exp(j2\pi f_c t) + \exp(j2\pi(-f_c)t)]/2$$

Two frequencies - one positive, one negative  
Each with half the amplitude of the original



$$X(f) \propto (\delta(f - f_c) + \delta(f + f_c))$$

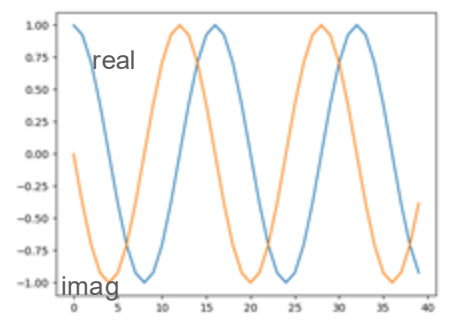
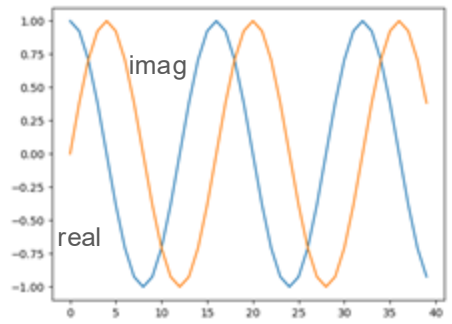
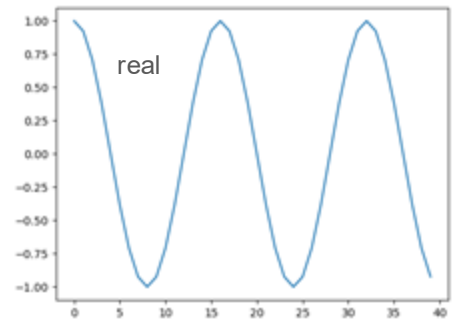
# Real vs. complex signal (time, frequency versions)

Real Cos 2000 Hz

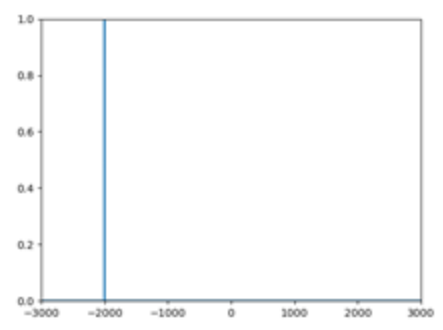
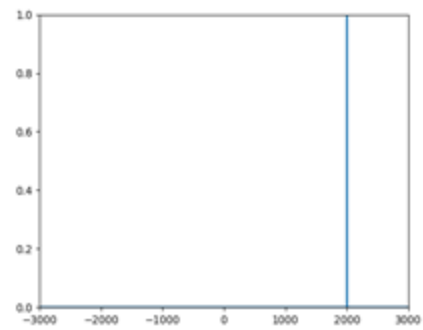
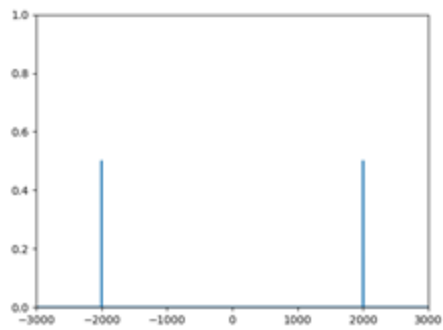
Complex IQ 2000 Hz

Complex IQ -2000Hz

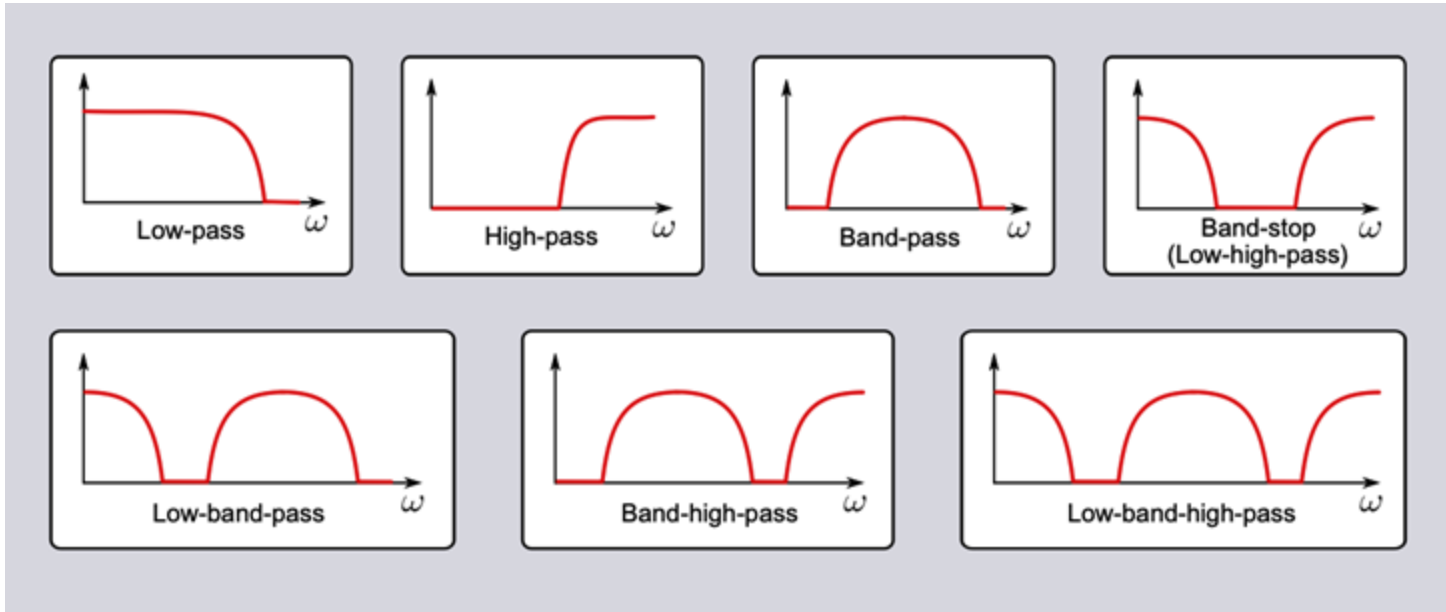
time



frequency



# Filters: Signal Transformations



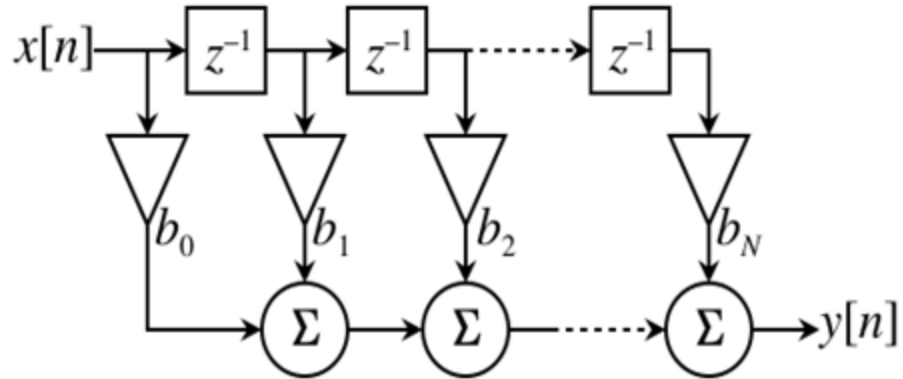
By  
SpinningSpark  
at English  
Wikipedia, CC  
BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=11313700>

Alters information / frequency content of a signal

Can be frequency domain or time domain: this tutorial uses time domain

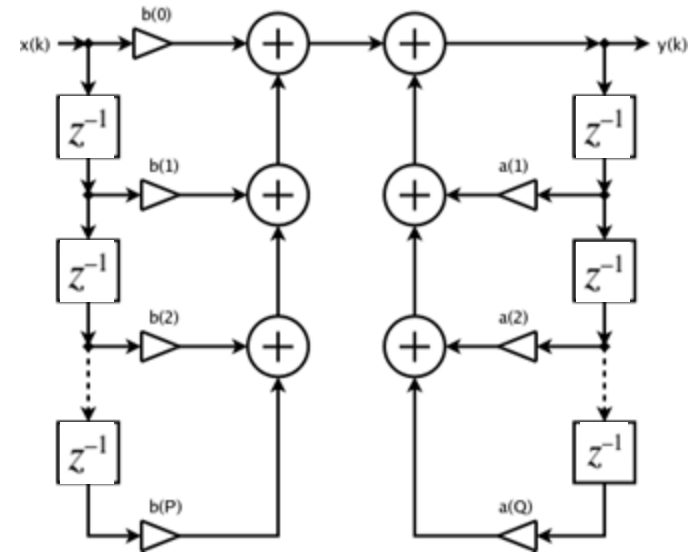
# Filter Types (Time Domain)

## Finite Impulse Response (FIR)



(derived from Wikipedia)

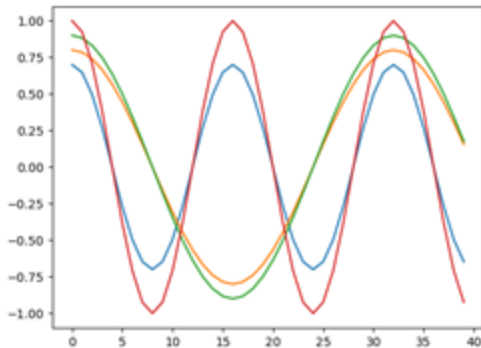
## Infinite Impulse Response (IIR)



# Common Elements for Filters Used Here

- All filters will be time domain filters, without use of Fourier Transforms.
- All filters will be finite impulse response (FIR) filters, except for the Notch filters which use infinite impulse response (IIR) filters.
- All filters will be applied to the same combined set of 4 input signals: -2000 Hz, -1000 Hz, 1000 Hz and 2000 Hz

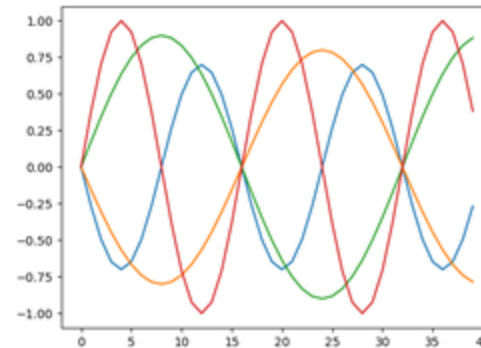
## Real



Python code to create IQ at single freq

```
delta_t = 1.0 / sample_rate
x = np.linspace(0.0, no_samps *
    delta_t, no_samps, endpoint=False)
signal_iq = amplitude *
    np.exp(1j * freq * 2.0 * np.pi * x)
```

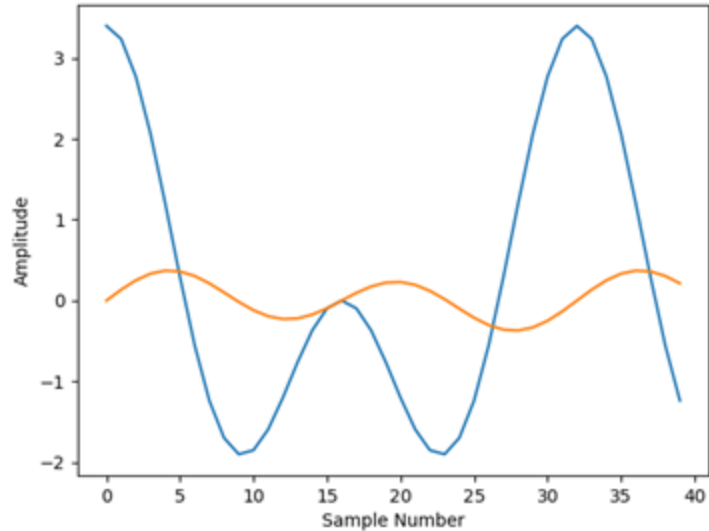
## Imag



Red 2 kHz    Green 1 kHz    Orange -1 kHz    Blue -2 kHz

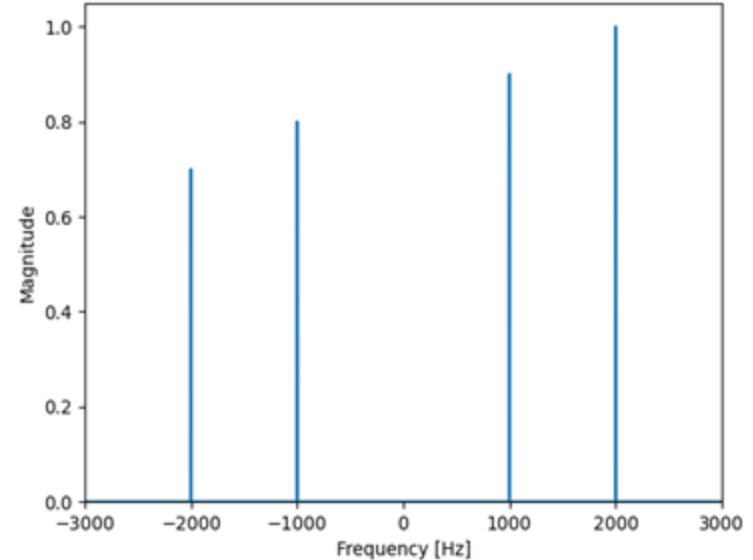
# Input Complex Baseband Signal: 4 frequencies combined

Time domain



Real (blue)  
Imag (orange)

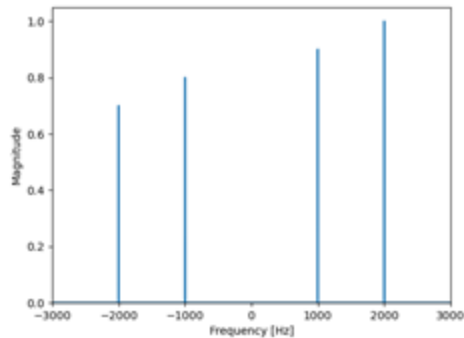
Frequency domain



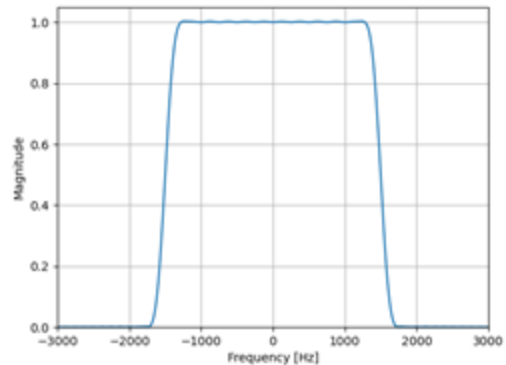
# Symmetric low pass filter (band pass filter centered at 0 Hz)

Frequency Domain

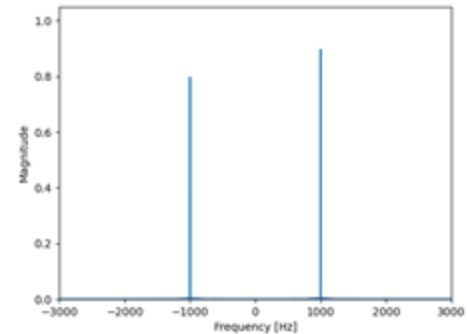
Input (sig)



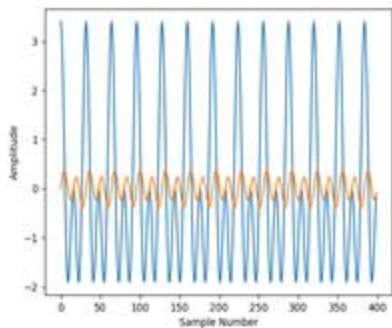
Filter amplitude response ( $|H|$ )



Output (y)



Time Domain



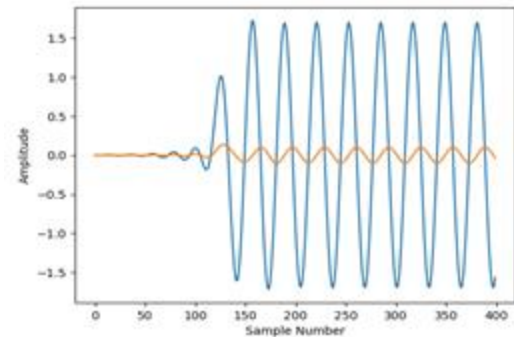
Python code:

```
h_lp = scipy.signal.firwin(num_taps,
f_lp_cutoff, fs=fs, pass_zero='lowpass')
```

```
a = [1.0] # filter is a FIR and not IIR
y = scipy.signal.lfilter(h_lp, a, sig)
```

```
w, H = scipy.signal.freqz(h_lp,
worN=2048, fs=fs, whole=True)
```

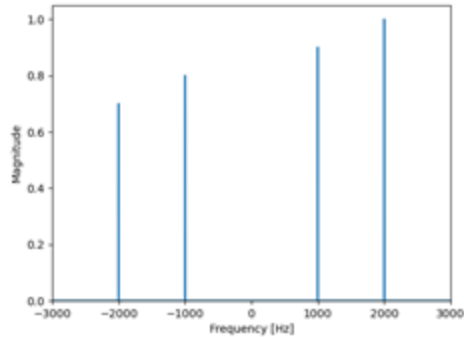
Real (blue)  
Imag (orange)



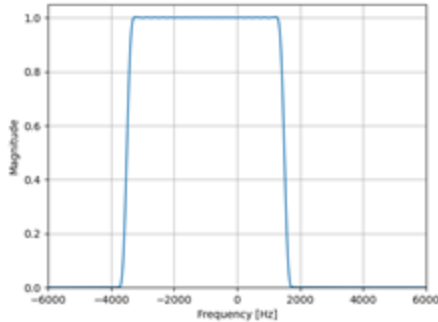
# Asymmetric band pass filter (filter centered at -1 kHz)

Frequency Domain

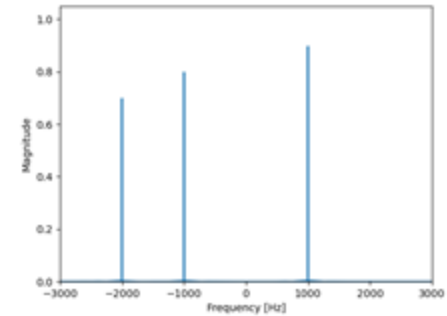
Input (sig)



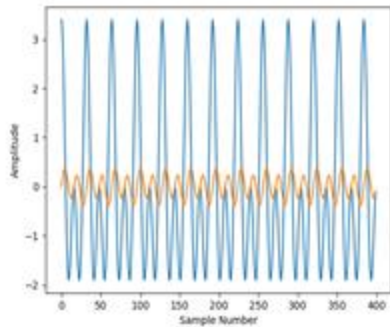
Filter amplitude response (|H|)



Output (y)



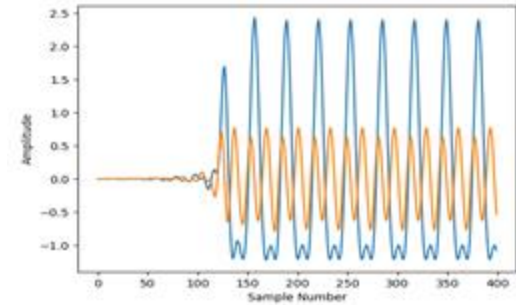
Time Domain



Python code:

```
h_lp = scipy.signal.firwin(num_taps,
    f_lp_cutoff, fs=fs, pass_zero='lowpass')
# Complex sinusoid at the center frequency
complex_shift = np.exp(1j * 2 * np.pi
    * f_center / fs * n)
# Shift the filter taps to create the
complex bandpass filter
h_bp_complex = h_lp * complex_shift
```

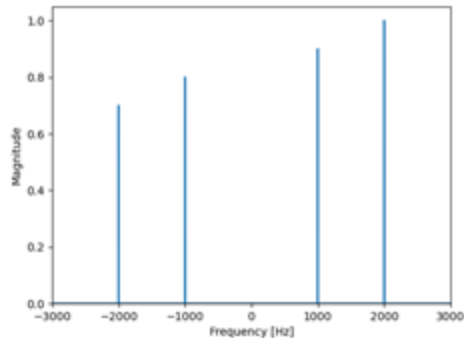
Real (blue)  
Imag (orange)



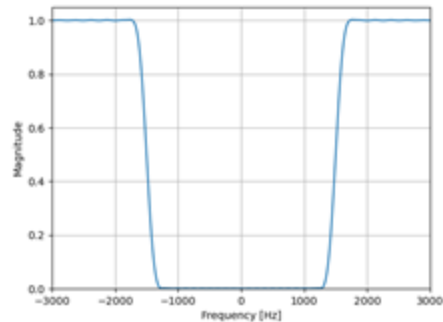
# Symmetric high pass filter (wide notch filter centered at 0 Hz)

Frequency Domain

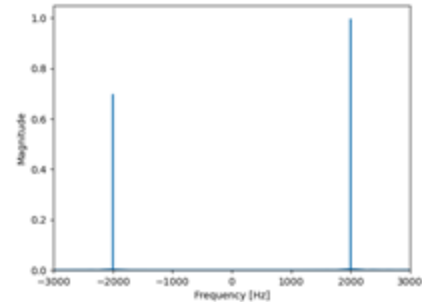
Input (sig)



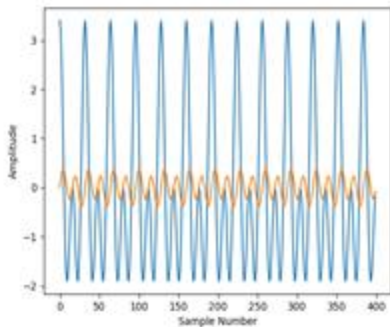
Filter amplitude response ( $|H|$ )



Output (y)



Time Domain



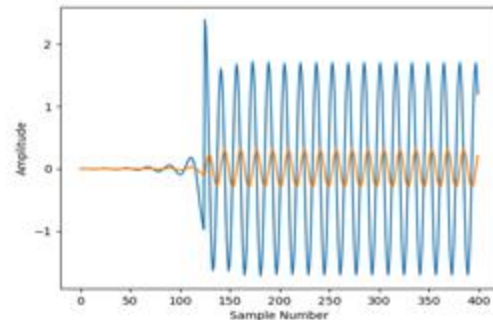
Python code:

```
h_hp = signal.firwin(num_taps, f_hp_cutoff,
                    fs=fs, pass_zero='highpass')
```

```
a = [1.0] # filter is a FIR and not IIR
y = scipy.signal.lfilter(h_hp, a, sig)
```

```
w, H = scipy.signal.freqz(h_hp,
                          worN=2048, fs=fs, whole=True)
```

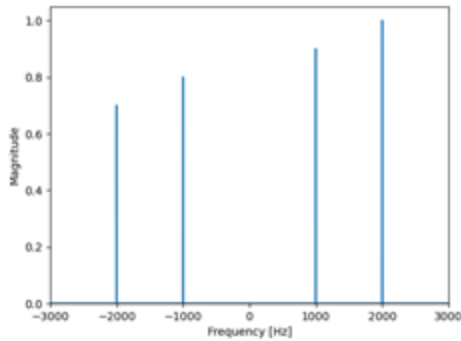
Real (blue)  
Imag (orange)



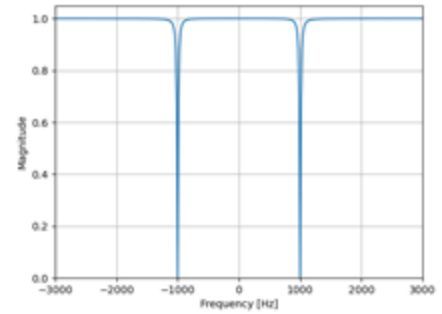
# Symmetric notch filter (notch defined at 1 kHz)

Frequency Domain

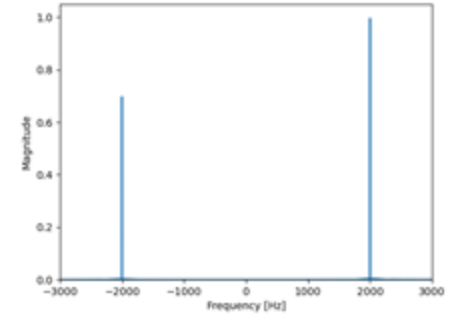
Input (sig)



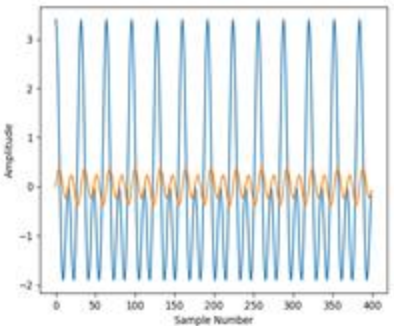
Filter amplitude response (|H|)



Output (y)



Time Domain

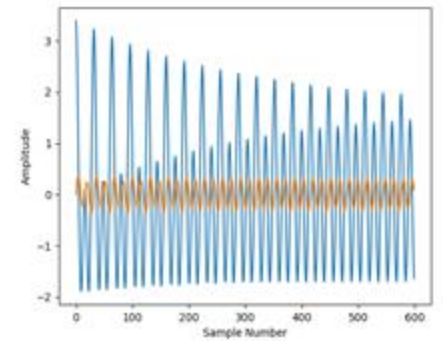


Python code

Real (blue)  
 Imag (orange)

```
# frequency of notch = 1000 Hz
b, a = signal.iirnotch(1000, quality, fs)

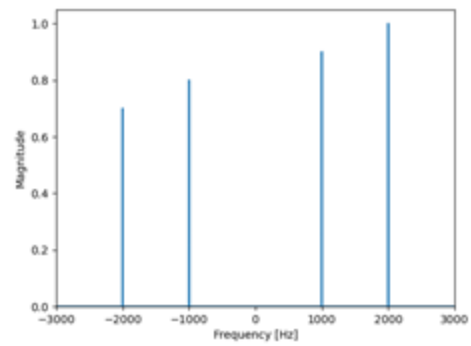
# apply the iir filter
y = signal.lfilter(b, a, signal_iq)
```



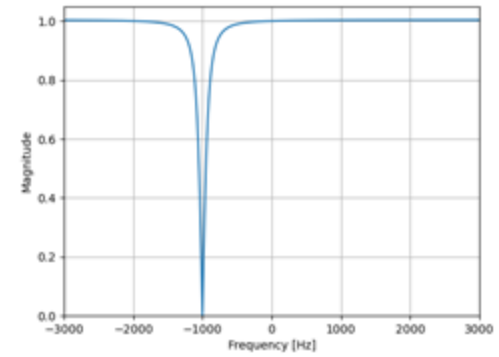
# Asymmetric notch filter (notch defined at -1 kHz)

Frequency Domain

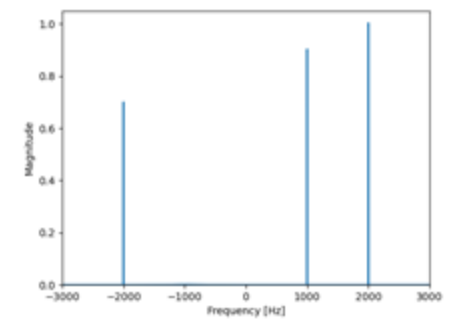
Input (sig)



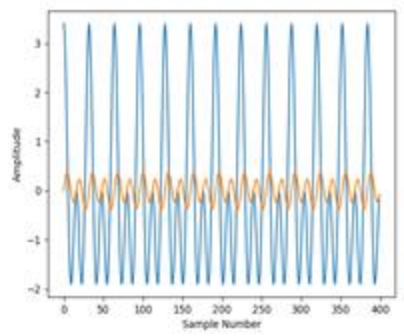
Filter amplitude response (|H|)



Output (y)



Time Domain



```
# Normalized frequency (-0.5 to 0.5)
w0 = notch_freq / sample_rate
```

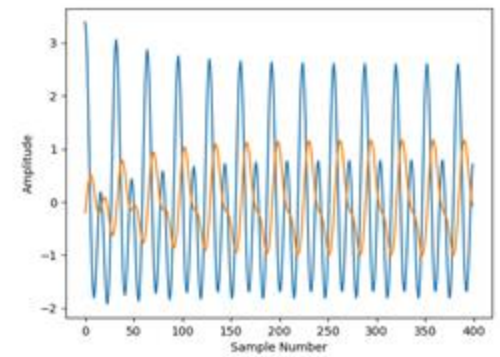
Real (blue)  
Imag (orange)

```
# Zero at notch frequency (on unit circle)
zero = np.exp(2j * np.pi * w0)
```

```
# Pole at same angle but inside unit circle (controlled by Q factor)
```

```
r = 1 - (1 / (2 * q_factor)) # Pole radius
pole = r * np.exp(2j * np.pi * w0)
```

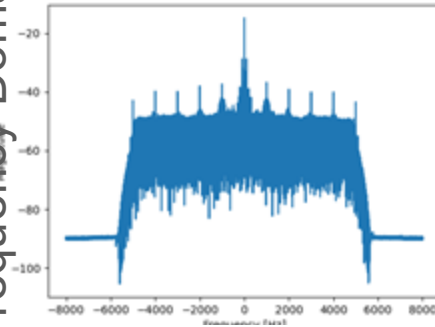
```
# Convert to transfer function coefficients
# H(z) = (z - zero) / (z - pole)
b = np.array([1, -zero])
a = np.array([1, -pole])
```



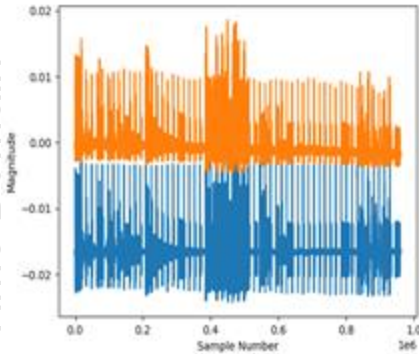
# AM Demodulation

Frequency Domain

Input (sig)



Time Domain



Real (blue)  
Imag (orange)

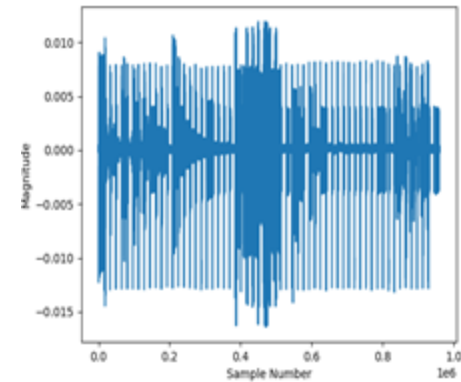
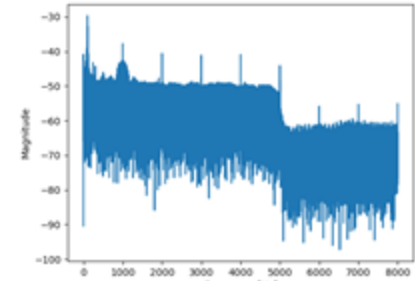


Python code

```
# signal_iq is the baseband iq array
```

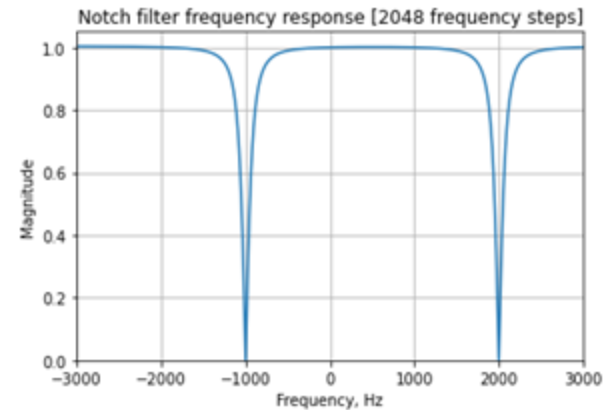
```
demod_am = np.abs(signal_iq)
demod_ave = np.average(demod_am)
demod_base = demod_am - demod_ave
```

Output audio (y)



# Demonstrations

- Jupyter notebooks available with baseband IQ signal and filter demonstrations
- Different filters and processing demonstrations - some not covered in this tutorial due to time restrictions
  - Symmetric and asymmetric filters
    - Low pass
    - Band pass
    - Single and multiple notch
  - AM demodulation
  - Narrowband FM demodulation
  - Mixing passband signals to baseband
  - Mixing baseband signals to passband
- Come find us at the Sunday demo tables!



Example: asymmetric notch filter  
Frequency response

## References

- PySDR: A Guide to SDR and DSP using Python, in particular the section on Filters  
<https://pysdr.org/>
- GnuRadio suggested reading  
<https://wiki.gnuradio.org/index.php/SuggestedReading>
- Radioconda (see Ryan Volz at this meeting)
- I/Q Data for Dummies  
<http://whiteboard.ping.se/SDR/IQ>
- Many online books and videos

## Resources

- Jupyter notebook repository at the HamSCI 2026 workshop github site
- Posted now at  
<https://github.com/HamSCI>
- Oppenheim & Schafer, Discrete-Time Signal Processing (3rd ed.)
- Many others

Acknowledgments: Paul Maine, KI5MIV, “Paul the SDR Guy”, for the audio file used for the FM demodulation; Python “scipy”, “matplotlib”, “numpy”, “sounddevice”, “wavinfo” libraries

# Acknowledgments

THE UNIVERSITY OF  
**SCRANTON**  
A JESUIT UNIVERSITY



The [HamSCI Community](#) is led by [The University of Scranton Department of Physics and Engineering W3USR](#), in collaboration with [Case Western Reserve University W8EDU](#), the [University of Alabama](#), the [New Jersey Institute of Technology Center for Solar Terrestrial Physics K2MFF](#), the [MIT Haystack Observatory](#), [TAPR](#), additional collaborating universities and institutions, and volunteer members of the [amateur radio](#) and citizen science communities.

We are grateful for the financial support of the [United States National Science Foundation](#), [NASA](#), and [Amateur Radio Digital Communications \(ARDC\)](#).

HamSCI silhouette photo by Ann Marie Rogalcheck-Frissell KC2KRQ.