SDR

Ben Matthews

NerdFest 2022
February 2022

Warning! Contains Math

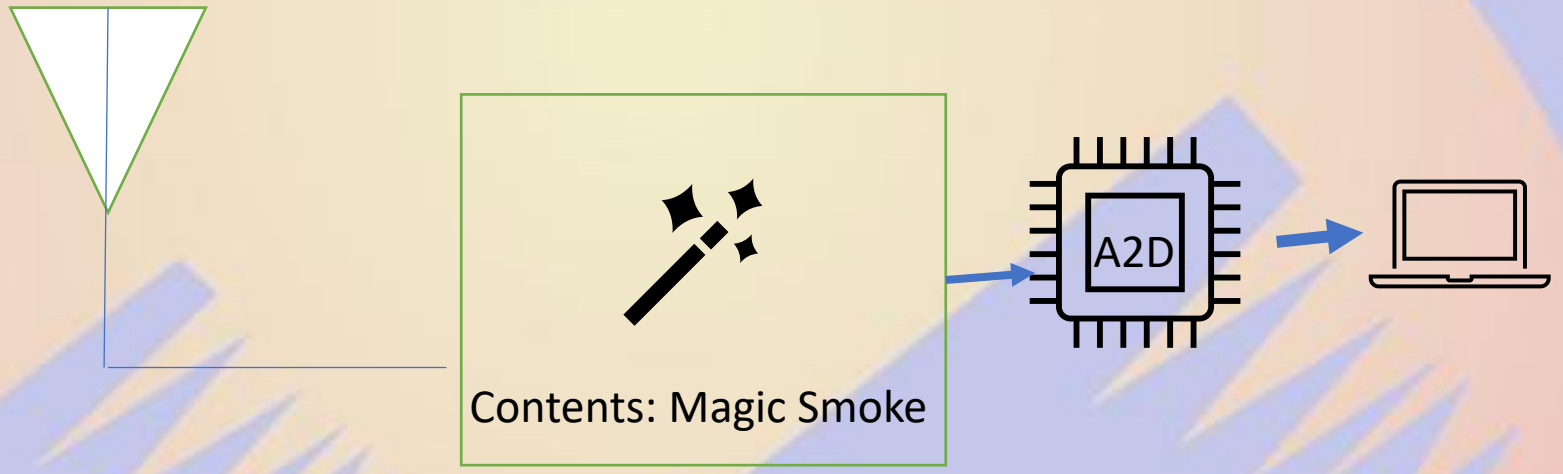
Disclaimer

- I’m going to try to show quite a bit of signal processing in half an hour
- So, it’s going to be pretty hand-wavey
- We might lose a factor of 2 or so in several places
  - But that’s what automatic gain is for ;-)
- The point is to understand how SDR works, not get the details right
- I’m not really a math person anyway

Definitions

- Radio: Device for converting magic to data
- Magic: Something I don’t care to explain
- SDR: Software defined radio
  - Make the computer do the magic
- Imaginary number: sqrt(-1) (aka magic)

Radio for the ~~Magician~~ Computer Nerd



SDR

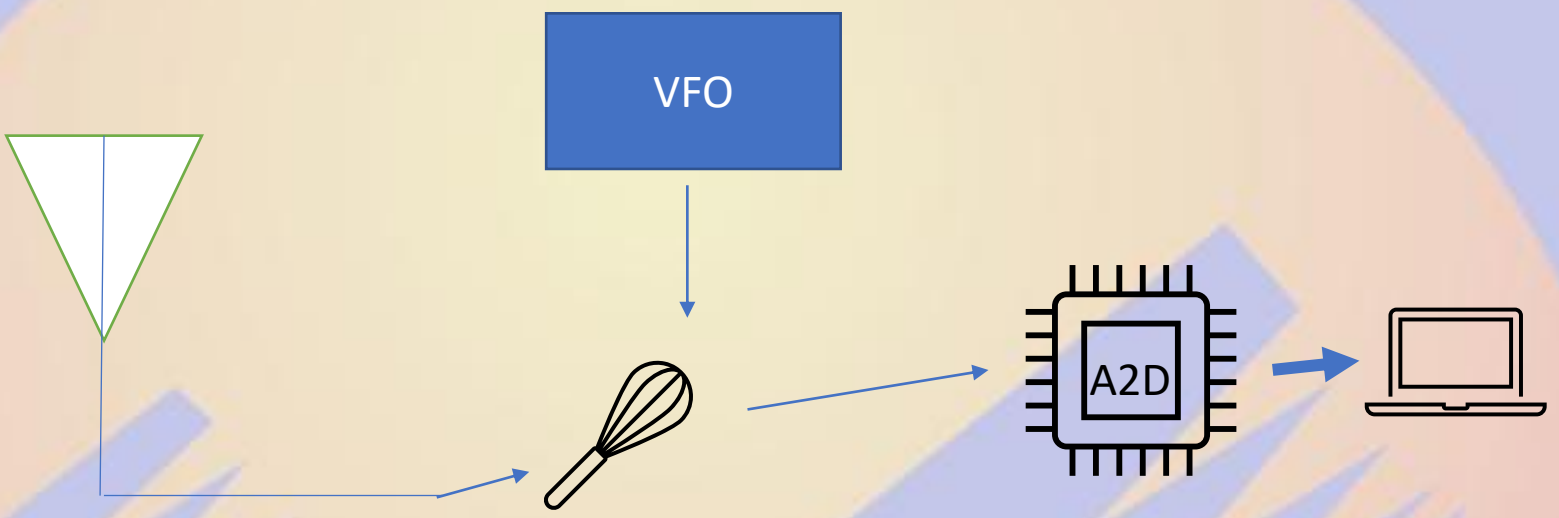
SDR

- Connect the antenna directly to the A2D
  - Ok, fine, ish. Maybe we want an amplifier too
- Need to sample the RF signal at twice the frequency of interest (or so says Nyquist)
- Fast A2D is expensive, so we'll allow a little magic – a mixer
- Let's only talk about receive for now. SDR Transmitters do exist too.

Real world aside: WebSDR

- You can buy an SDR that'll capture the entire HF spectrum
- Put it on the web and let people process as they see fit
- KiwiSDR - ~$300
- Listen to multiple stations at once (or do digital decoding or whatever)
- Web based UI. Some are even publicly available
  - Rx.kiwisdr.com

SDR



Sounds cool, how?

Amazon.com: NESDR SMArTee

https://www.amazon.com/NooElec-NESDR-SMArTee-Bundle-R820T2-Based/dp/B079C4S2BT/ref=sr_1

All | Best Sellers | Amazon Basics | New Releases | Customer Service | Today's Deals | Prime | Books | Music | Black is remarkable

Computers | Laptops | Desktops | Monitors | Tablets | Computer Accessories | PC Components | PC Gaming | Deals

Electronics › Computers & Accessories › Computer Components › External Components › External TV Tuners

Sponsored

Roll over image to zoom in

NESDR SMArTee v2 Bundle - Premium RTL-SDR w/Integrated Bias Tee, Aluminum Enclosure, 0.5PPM TCXO, SMA Input, Antenna Base & 3 Antennas. RTL2832U & R820T2-Based Software Defined Radio (SDR)

Brand: NooElec

436 ratings | 24 answered questions

$41.95

& FREE Returns

Get $60 off instantly: Pay $0.00 ~~$41.95~~ upon approval for the Amazon Prime Store Card. No annual fee.

$41.95

& FREE Returns

FREE delivery **February 21 - March 1**

Or fastest delivery **February 15 - 24**

Select delivery location

Quantity: 1

Add to Cart

Buy Now

Secure transaction

RTLSDR

- Chip designed to capture European Digital TV
- Can be tuned *way* out of band
  - ~DC to 1GHz
- Let's assume you have a Linux computer (Pi is fine). Windows people are on their own
- Since it's really a TV tuner, we need to disable the TV drivers…

RTLSDR

Put the following in /etc/modprobe.d/blacklist-rtlsdr.conf and reboot:

blacklist rtl2832

blacklist dvb_usb_rtl28xxu

blacklist rtl2832_sdr

blacklist rtl8xxxu

RTLSDR

- Let’s install some software:
- apt-get install rtlsdr-tools soapysdr-tools soapysdr-module-all
- For those who just want a reciever, I like “gqrx”
- For those of you not afraid of a little math, I’ll be using the “Julia” programming language, but it’s just math. Use whatever. Let’s write a receiver!

…but first: Trigonometry

- Signal = sin(2*pi*f*t)
  - f = frequency
  - t = time
- sin(a)sin(b)=1/2(cos(a-b) – cos(a+b))
  - **We'll use this for implementing our tuner**
- sin(x) = cos(x+pi/2)

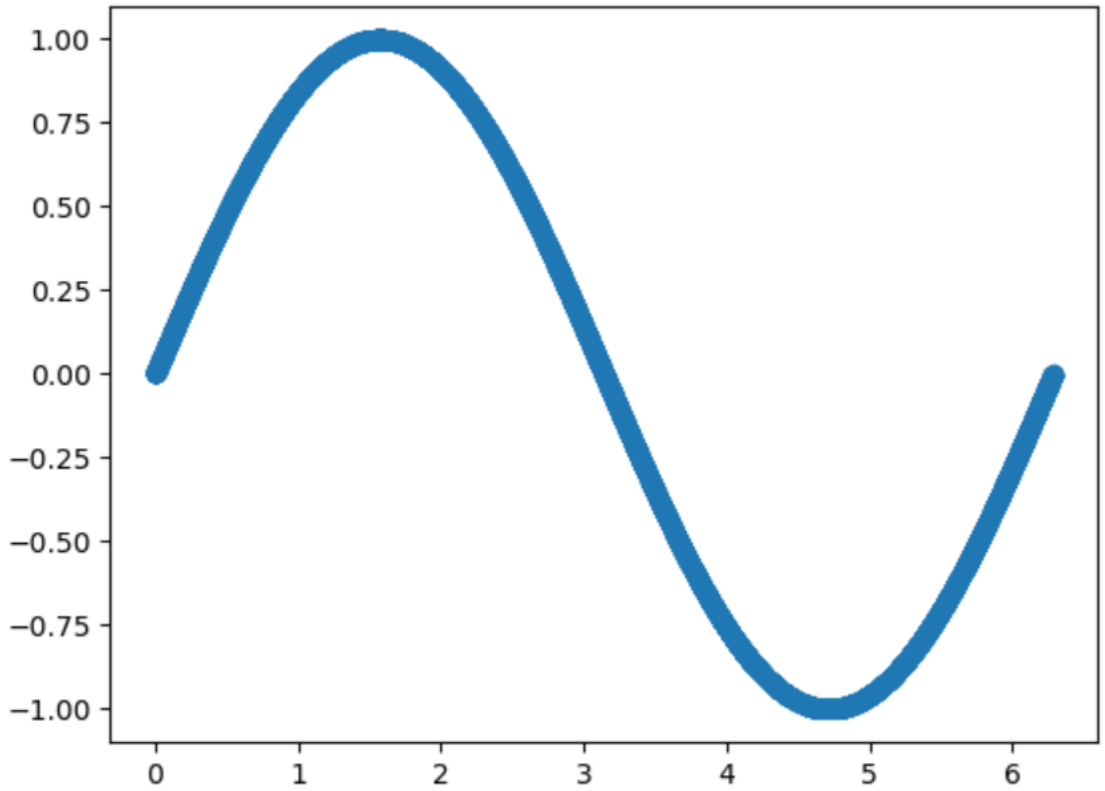
… but Radio is in Frequencies, right?

- If you insist. Fourier says we can represent any signal as a sum of (probably infinitely many) sine waves
- We can convert between the time domain (signal(time)) and the frequency domain (signal(frequency)) using the Fourier transform
- Fourier transforms are magic. Refer to your favorite signals textbook. I'm just going to use FFTW
  - "Fastest Fourier Transform in the West" (software library)
    - Yes, there's an FFTE. Computer people are silly.

Visual Learners

In [15]:
```
using PyPlot, FFTW, DSP
```

In [39]:
```
samples = zeros(10000)
frequency = 1
for t in 1:10000
    samples[t]=(2*pi)*frequency*1/10000*t
end
sine = sin.(samples)
scatter(samples,sine)
```

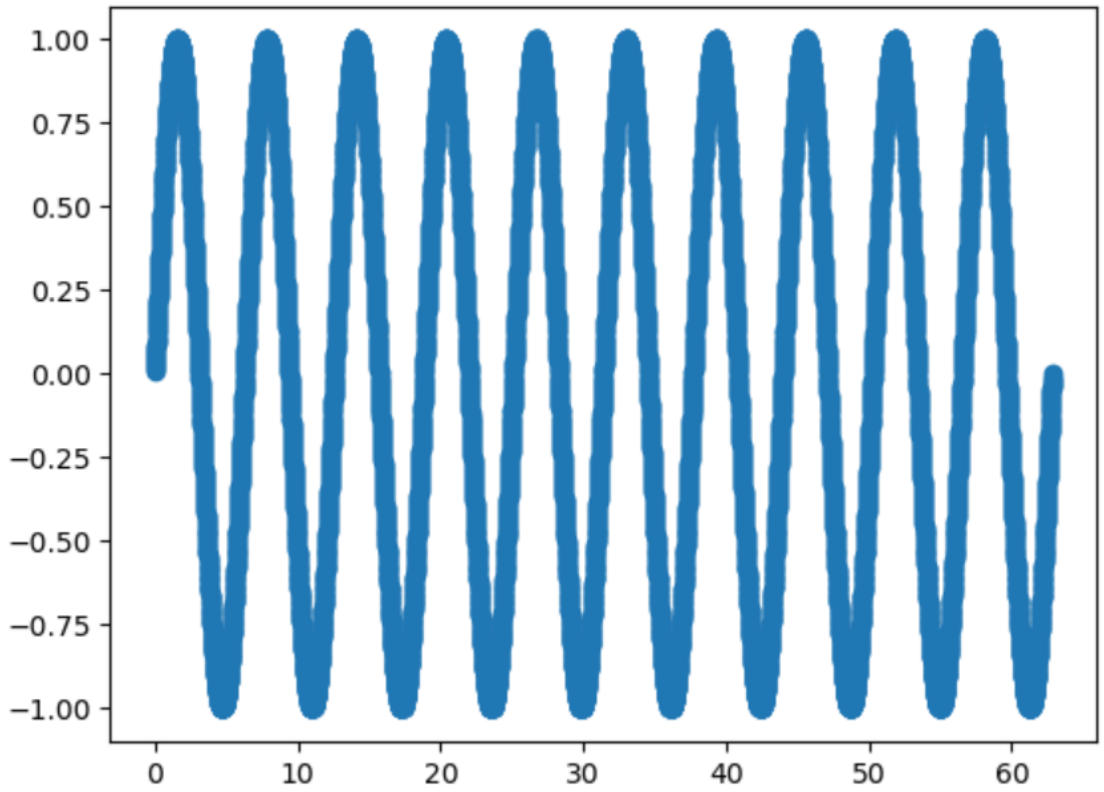


Out[39]: PyObject <matplotlib.collections.PathCollection object at 0x7fdc8ce932b0>

Increase the Frequency?

In [51]:
```
using PyPlot, FFTW, DSP
```

In [43]:
```
samples = zeros(10000)
frequency = 10
for t in 1:10000
    samples[t]=(2*pi)*frequency*1/10000*t
end
sine = sin.(samples)
scatter(samples,sine)
```

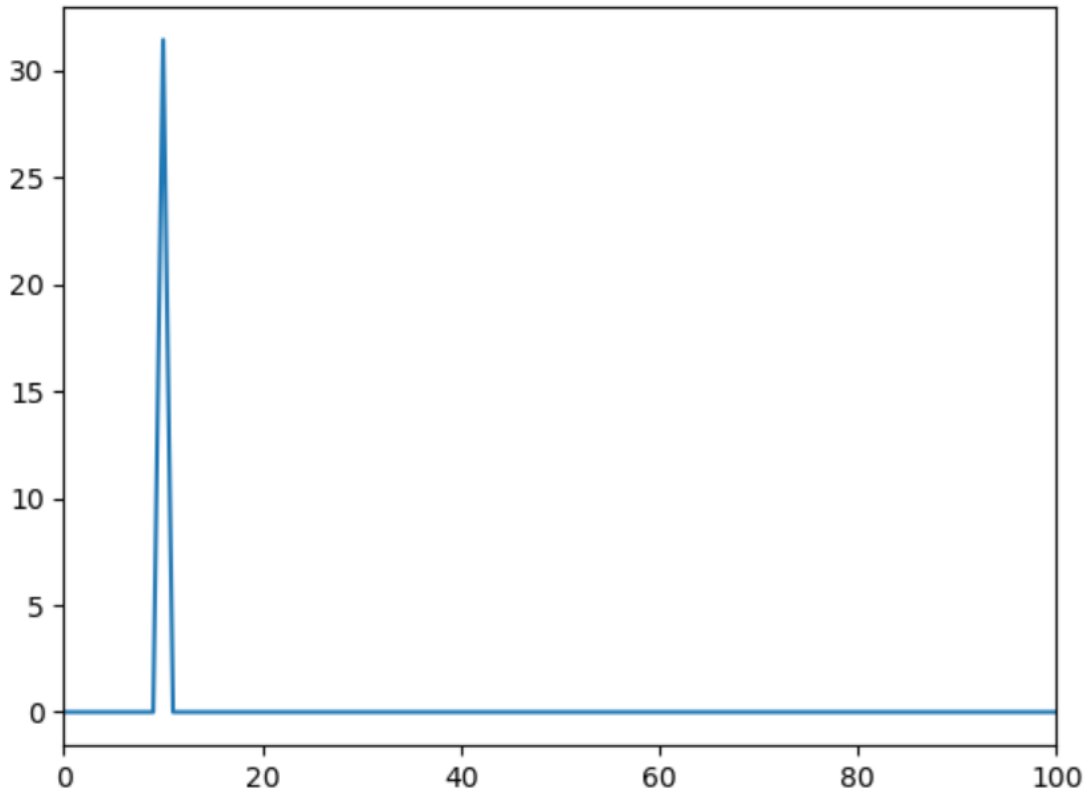


Out[43]: PyObject <matplotlib.collections.PathCollection object at 0x7fdc8cdea7f0>

Frequency Domain

In [55]:

```
plot(fft(sine))
ax = gca()
ax[:set_xlim]([0, 100])
```

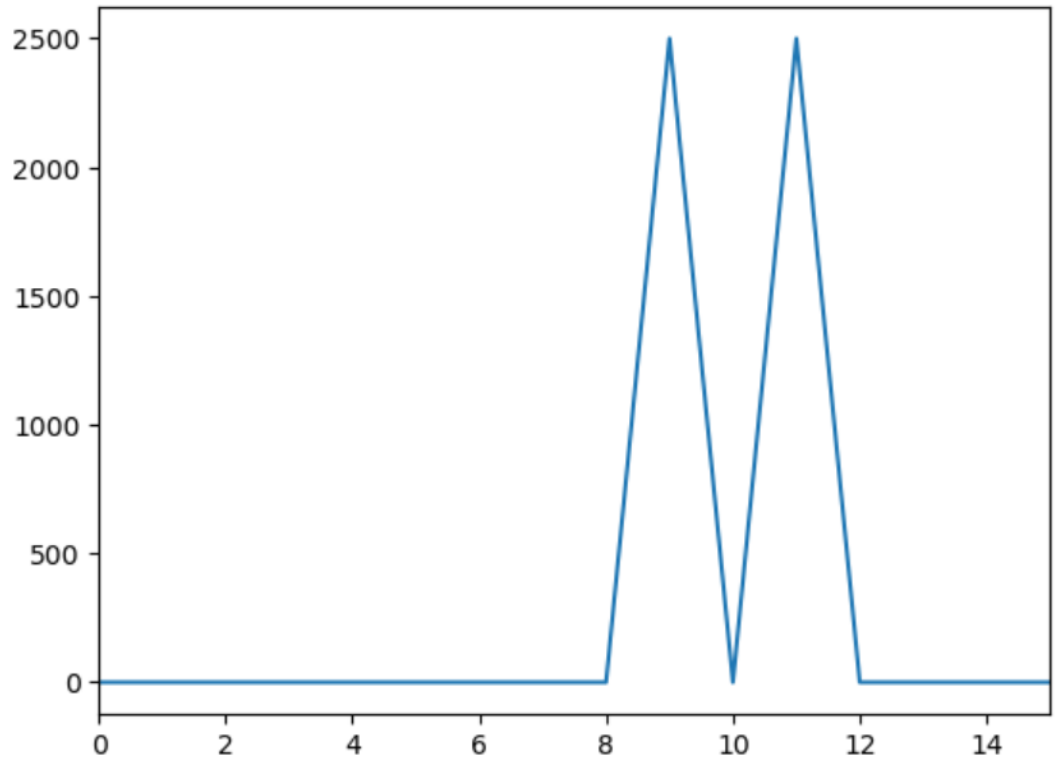



```
/home/matthb2/.julia/conda/3/lib/python3.9/site-packages/matplotlib/cbook/__init__.py:1298:
ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

Mixing a f=10 and f=1 signals (frequency domain)

In [60]:

```
ften = sin.(samples*10)
fone = sin.(samples*1)
mixed = ften .* fone
plot(abs.(fft(mixed)))
ax = gca()
ax[:set_xlim]([0,15])
```



Out[60]: (0.0, 15.0)

In [ ]:

Back to SDR: Let's capture some samples

rx_sdr -f 144000000 -d driver=remote -s 6144000 -n 61440000 test_wide.iq

- Give me samples centered around 144MHz
- Sample at ~6MHz
- Give me 61440000 samples (10 seconds)
- Default format (8 bit/sample, complex, interleaved I/Q)

I/Q?

- Signal = I*cos(2*pi*f*t)+Q*sin(2*pi*f*t)
- I’m told this makes things easier
- Explaining why is probably too much for this talk
- Let’s just map our samples into complex floats and not think about it too much for now.

What did our SDR give us exactly?

- Integer samples
- High bit is a sign
- Everything is “baseband”. Basically, centered around our specified center frequency. f=0 is our center or “dial” frequency
  - This means we have negative frequencies.. Just go with it – the math works anyway.
- We need to keep track of dial frequency/sample rate ourselves

Let’s read in our samples

In [18]:
```
struct UInt8Sample
    I::UInt8
    Q::UInt8
end
```

In [35]:
```
samplerate = 6144000
centerfreq = 144000000
channeldial = 146540000
```

Out[35]: 146540000

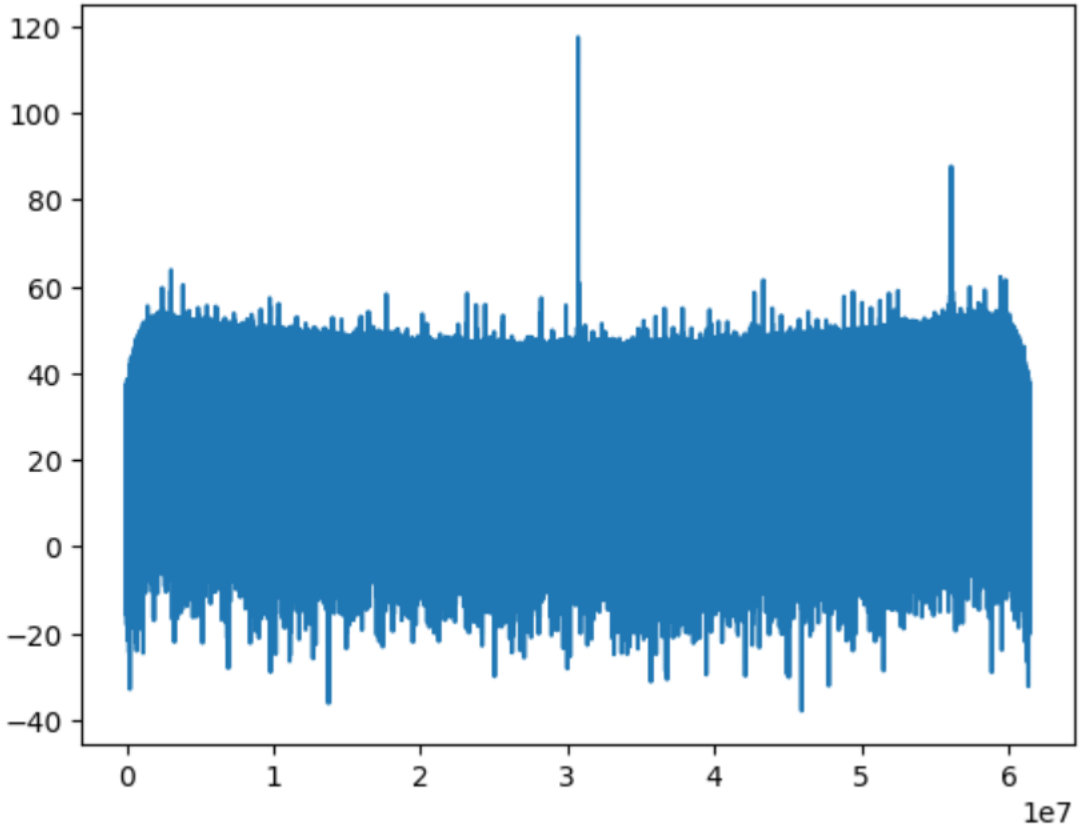
In [24]:
```
file = "test_wide.iq"
sz = stat(file).size÷2
samples = Array{UInt8Sample}(undef, sz)
fd = open(file, "r")
read!(fd, samples)
close(fd)
length(samples)
```

Out[24]: 61440000

In [25]:
```
cmplxsamples = Array{Complex{Float32}}(undef,length(samples))
for i in 1:length(samples)
    cmplxsamples[i] = samples[i].I/(128.0)-1.0 + im*(samples[i].Q/(128.0)-1.0)
end
```

A first look at our data (frequency domain)

```
In [33]: plot(20 .*log10.(abs.(fftshift(fft((cmplxsamples))))))
```




```
Out[33]: 1-element Vector{PyCall.PyObject}:
          PyObject <matplotlib.lines.Line2D object at 0x7f580fe90e50>
```

Tuning to the frequency of interest

- Compute a signal with a frequency equal to the amount we want to shift
  - That is, the frequency difference between our dial frequency and the offset of the signal we're interested in
  - Remember: We're working in I/Q
- Multiply our signal by this intermediate frequency.

Tuning to the frequency of interest

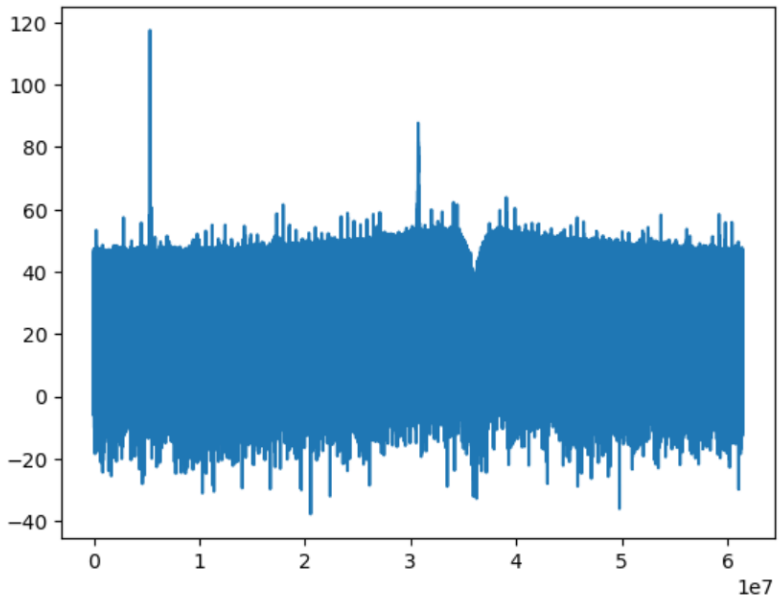
In [41]:
```
shift_step = ((channeldial-centerfreq)/samplerate)*2*pi #shift per sample
```

Out[41]: 2.5975408008196856

In [46]:
```
shift = shift_step
for i in 1:length(cmplxsamples)
    shifted[i] = cmplxsamples[i] * (sin(shift) + cos(shift)*im)
    shift += shift_step
    while shift > 2*pi
        shift -= 2*pi
    end
end
0
```

Out[46]: 0

In [47]:
```
plot(20 .*log10.(abs.(fftshift(fft((shifted))))))
```



Performance: Decimation

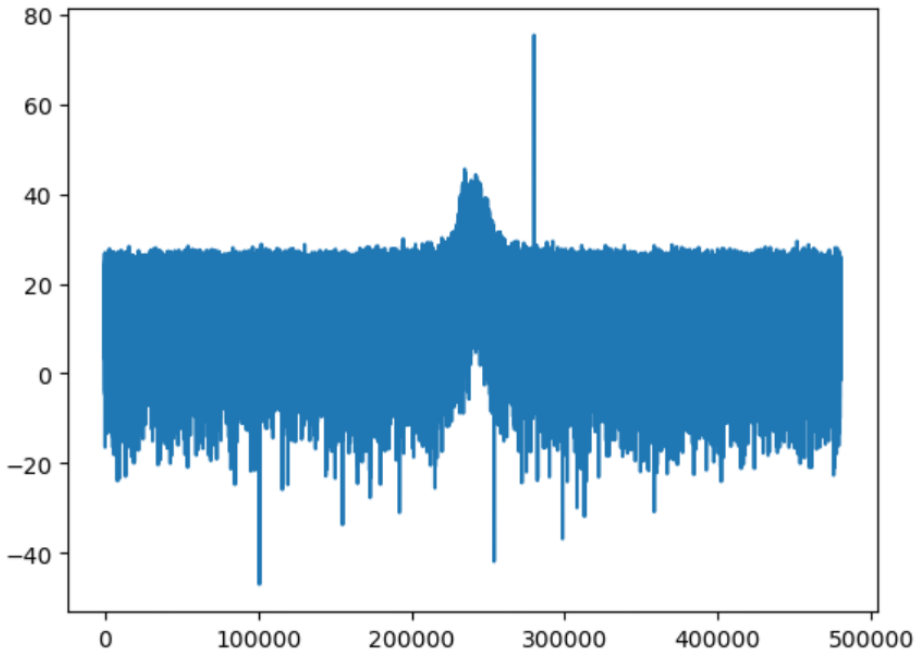
- We have a *lot* of data. 6 Msamples/sec
- We only need to represent ~25 KHz and the signal we're interested in is centered at zero
- To represent 25KHz, we need ~2x that. Let's keep 48KHz since audio libraries like to work at 48KHz sample rates
- So, we can throw away most of the data without any important loss
  - Unless we want to retune to a different part of the signal

Decimation

In [20]:
```
#This happens to work out evenly. Could be more complicated
decimate_factor = samplerate÷48000
decimatedsamples = Array{Complex{Float32}}(undef,length(shifted)÷decimate_factor)
i=1
j=1
while i<length(shifted)
    decimatedsamples[j] = shifted[i]
    i+=decimate_factor
    j+=1
end
length(decimatedsamples)
```

Out[20]: 480000

In [21]:
```
plot(20 .*log10.(abs.(fftshift(fft((decimatedsamples))))))
```



Out[21]:
```
1-element Vector{PyCall.PyObject}:
 PyObject <matplotlib.lines.Line2D object at 0x7fd14abfad30>
```

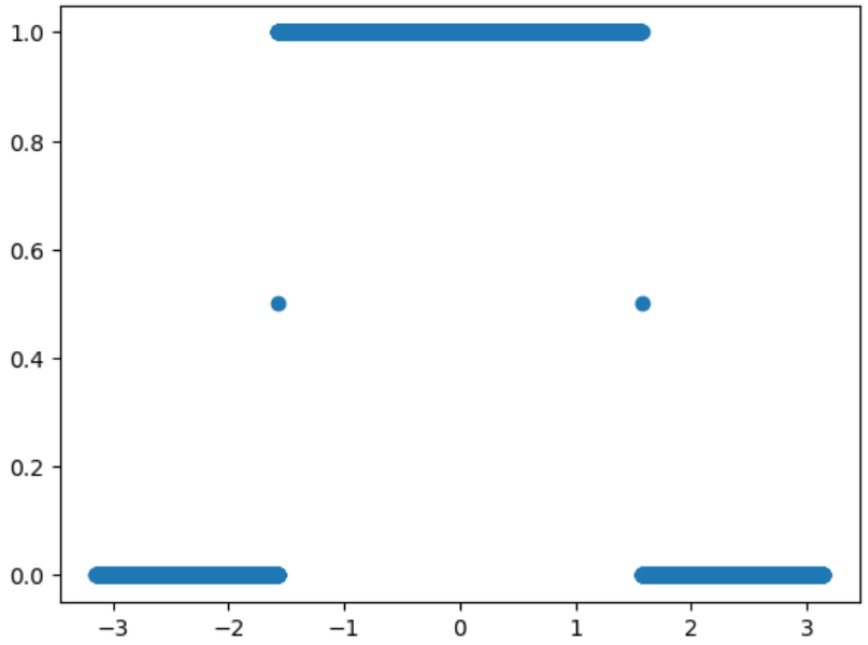
Filtering

- We want what radio folks call a lowpass filter. Half our signal is on either side of zero, so a low pass about zero is appropriate
- You could also think of it as a bandpass around our signal
- Lots of ways to do this, but let's talk about a FIR filter (Finite Impulse Response)

Lowpass Filter

In [75]:

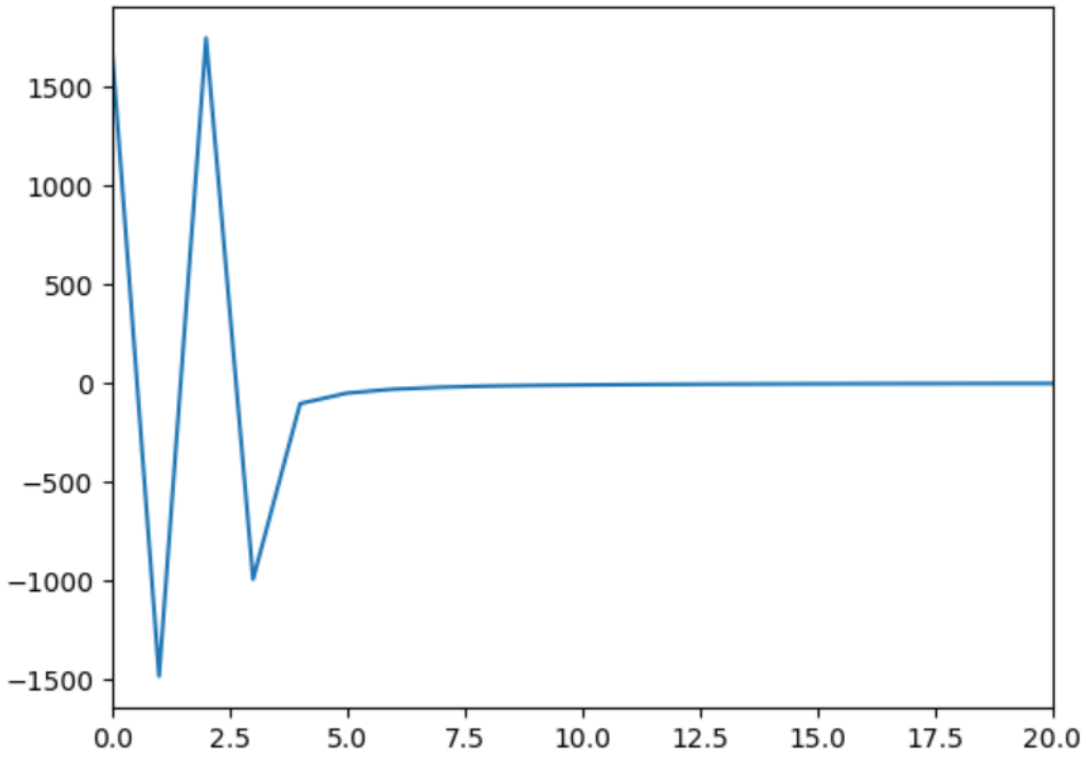
```
times = zeros(10000)
rectsamples = zeros(10000)
#this is often called a rectangle function
for t in 1:10000
    times[t]=((2*pi)*1*1/10000*t)-pi #subtract pi to shift us to both sides of zero
    if abs(times[t]) > pi/2
        rectsamples[t] = 0
    end
    if abs(times[t]) == pi/2
        rectsamples[t] = .5
    end
    if abs(times[t]) < pi/2
        rectsamples[t] = 1
    end
end
scatter(times, rectsamples)
```



Ifft(lowpass)

In [85]:

```
plot(fft(real.(rectsamples)))
ax = gca() #I'm cheating here and only showing you the low frequency components
ax[:set_xlim]([0, 20])
```



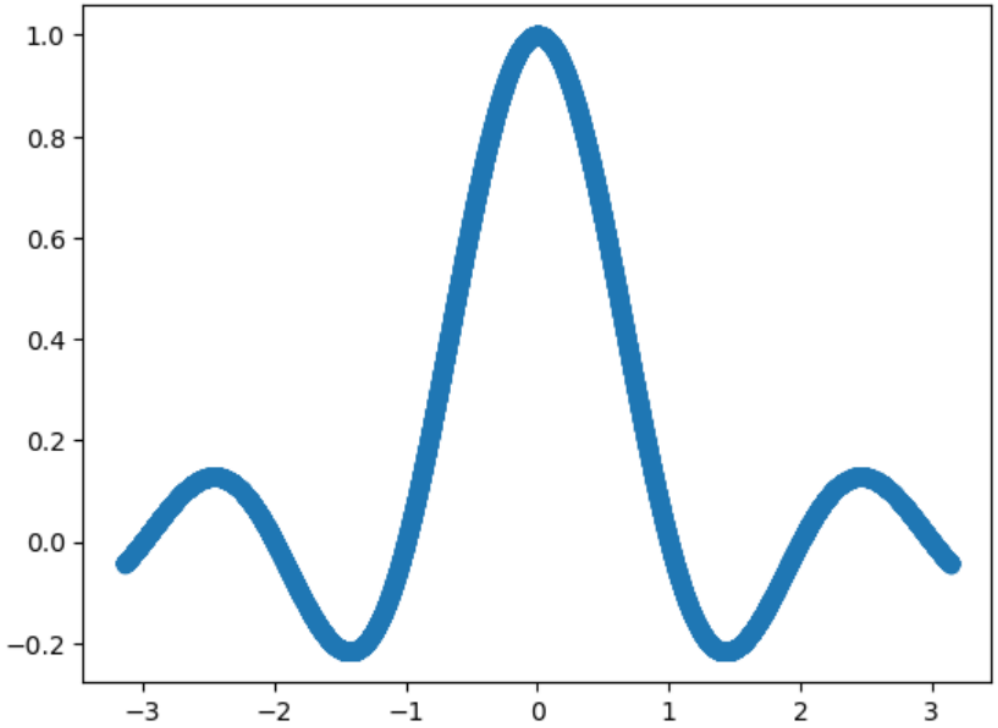

```
/home/matthb2/.julia/conda/3/lib/python3.9/site-packages/matplotlib/cbook/__init__.py:1298:
ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

Ifft(lowpass) (higher precision) (yes, I cheated and looked it up)

In [80]:

```
times = zeros(10000)
rectsamples = zeros(10000)

for t in 1:10000
    times[t]=((2*pi)*1*1/10000*t)-pi #subtract pi to shift us to both sides of zero
end
rectsamples = sinc.(times)
scatter(times, rectsamples)
```



Out[80]: PyObject <matplotlib.collections.PathCollection object at 0x7fdc72eed7f0>

Can I get that as a formula?

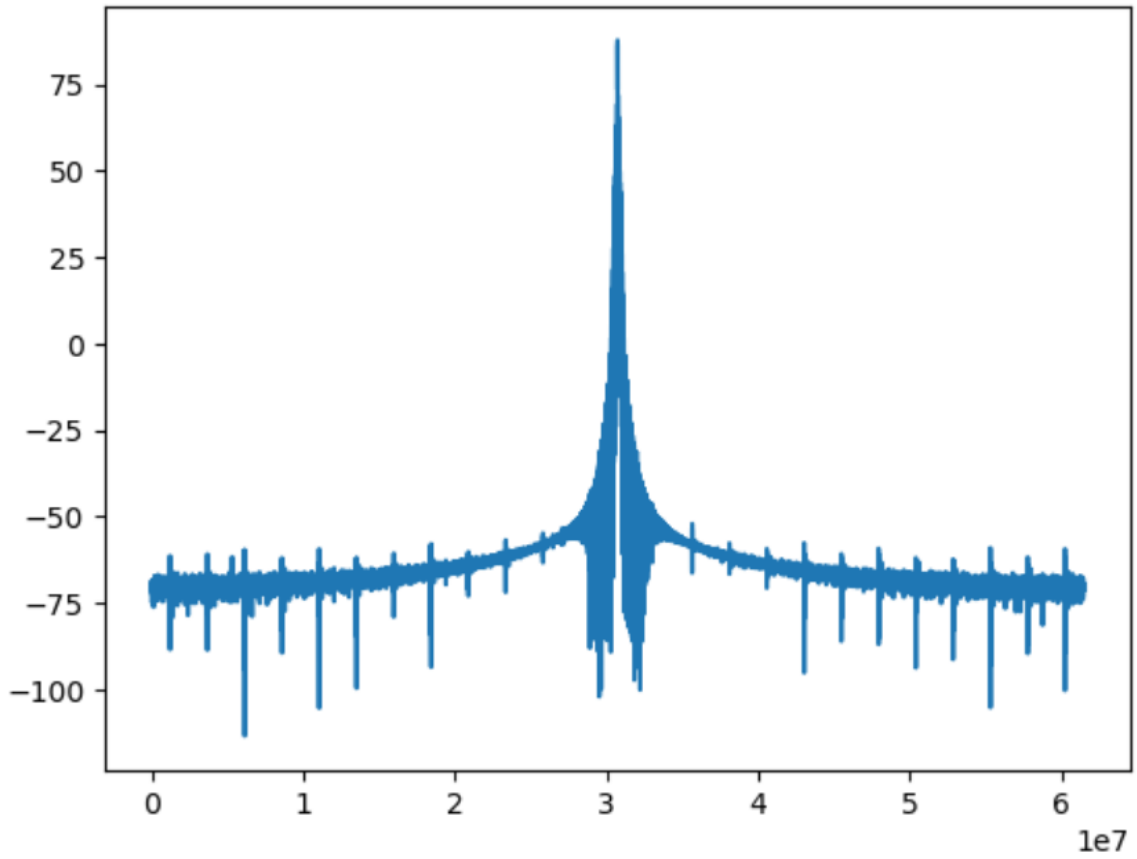
Tap[i] = Sin(2*pi*fc*i)/i

(fc = cutoff frequency)

- We just compute some taps and multiply them by our signal
- More taps = shaper cutoff and better attenuation, but also more computation
  - There are various ways of estimating how many taps are needed for particular filter properties. I'm just going to cheat and let the library worry about it.

Filtered signal (frequency domain)

```
In [136]: plot(20 .*log10.(abs.(fftshift(fft((out))))))
```




```
Out[136]: 1-element Vector{PyCall.PyObject}:
           PyObject <matplotlib.lines.Line2D object at 0x7fe3f4f360d0>
```

Demodulation

- For AM, we're pretty much done. Send the samples to the sound card.
- For SSB, we'd use a slightly different filter to reject one of the sidebands
- For FM, we need to look at how the frequency/phase changes between samples
- For digital, well, there are lots of schemes.

Other things we’re not talking about

- Gain/Volume control
- Extra DSP to improve audio
- Squelch

Why would I want to do this?

- Play with new modes
- Cheaper than buying a radio for each protocol/mode
- Decode something interesting
- Decode all the signals in a band with one antenna/radio
- Cheaper than building hardware to do it (in general)
- This talk was motivated by TDOA
- See what's out there on the airwaves

Application: TDOA

- If we know exactly when a signal is received at multiple sites, we can figure out where the transmitter is relative to those sites
- But we'd need to have perfectly synchronized clocks at each site
- Can we synchronize time based on when a known signal is received at known locations?
- I think yes, but this is a work in progress

But I’m not a programmer?

- GNU Radio has a flow chart based “programming” environment with “blocks” that do all these things that can be wired together.
- It does hide the details a little, but you can still demodulate pretty much anything
- https://www.gnuradio.org/

Questions?

ben@kc2vjw.com

References/Places to Look for More

Similar talk by HA7ILM: https://www.youtube.com/watch?v=-QERqK1XAy0

Free book from Mathworks: https://www.mathworks.com/campaigns/offers/download-rtl-sdr-ebook.html

PySDR Guide (useful even if you're not using Python): https://pysdr.org/index.html

Programming environment used to generate some of these slides: https://jupyter.org/ https://julialang.org/

More about GNURadio: https://wiki.gnuradio.org/index.php?title=Tutorials

Interesting SDR related news: https://www.rtl-sdr.com/