LAB 1 – January 22, 2019
EXPLORING DIGITAL SAMPLING, FOURIER TRANSFORMS, and both DSB and SSB MIXERS

Contents

The purpose of this lab is to experimentally investigate digital sampling, discrete Fourier Transforms, and mixers. Mixers are the basis of heterodyne spectroscopy. Heterodyne spectroscopy which, whether you knew it or not, is what you use every day that you listen to a radio, use a cell phone, watch TV—or do radio astronomy. In the second lab, we'll use it to observe the 21-cm line emission from hydrogen our Galaxy.

In this lab, you will be performing several experiments, analyzing the data, and generating a number of different data files. Keep careful notes, preferably in a **lab notebook**. (And no, a `jupyter` notebook is not equivalent.)

1. GOALS

- Learn how to sample electronic signals and convert them into digital signals.
- Get started with our programming language, Python, using it for the mathematical analysis, signal processing, and making nice plots.

- Become acquainted with aliasing and the basic law of sampling: the Nyquist criterion.
- Learn how to use Discrete Fourier Transforms (DFTs) to determine the frequency power spectrum of a time series. Understand leakage power and frequency resolution when sampling a single sine wave.
- Learn about the Fast Fourier Transform (FFT) as a fast implementation of the DFT.
- Learn how the FT treats noise, which is the important case for radio astronomy.
- Learn about the convolution theorem (and its cousin, the correlation theorem), and discover the relationship between autocorrelation functions and power spectra.
- Learn the basics of mixing for frequency conversion (the heterodyne technique). Explore how real mixers differ from ideal ones.
- Learn about negative frequencies and how complex inputs to a FT break the positive/negative frequency degeneracy.
- Construct a single-sideband (SSB) mixer and explore the mixing process.
- Learn enough LaTeXto write up your results in a formal lab report, including nice plots and graphs.

2. SCHEDULE

There's a lot to do in this lab! If you don't understand the Nyquist criterion by the end of the first week, you're behind. Here's how it should be:

1. *The First Week.* Finish §4, which requires reading the accompanying material in §5. *Be prepared to show your work, your software, and your results to the class, making real-time plots in Python during your presentation.*
2. *The Second Week.* Finish §6 and the reading in §7. Again, be prepared to strut your stuff to the class.
3. *The Third Week.* Read the handouts in §8, and then write and hand in your formal report! Your report should contain relevant plots together with commentary to illustrate your work, your thought processes, and your conclusions. Generally speaking, your lab report should address, with discussion and/or plots, each of the goals in §1.

3. SOFTWARE ENGINEERING

You are about to write a bunch of code. For many of you, this lab may require more coding than you've done before. As the scale of your programming grows, you will need to learn to organize, document, test, and stabilize your code. The process of building sound code is broadly referred to as **software engineering**, and in addition to being helpful for scientific research, it may be one of the most marketable skills you learn in this class.

Learning software engineering will be an on-going process, but let's start with a few principles for your first lab.

- *Package your code.* For now, this is means keeping the bulk of your code together in a directory that is separate from the data you acquire. You might also think about separating the **functions** and **classes** you build from the **scripts** and **notebooks** that execute that code. To do this, you'll want to build a **module**. Try it; it's easy!
- *Code twice.* Code once to figure out how to make it work, and then, once you know how it works and what you want to do, code it again, this time doing it right.
- *Revision control.* `git` is quickly becoming foundational to serious software engineering. Do youself a favor and open a GitHub account (they're free), make a project in your account for this class, and regularly commit to it. It backs up your work, allows you to make changes with impunity (you can always go back to a working state), and allows you to easily synchronize your work across multiple computers. Seriously, it's worth the learning curve. Just do it.

4. IN THE LAB: DIGITALLY SAMPLING A SINGLE SINE WAVE (First Week)

4.1. Handouts and Software

As you begin real work this first week, you will need to become immersed in the Linux operating system, your (Vi/Emacs/3rd party) text editor, and Python. To this end, you'll need to become familiar with the resources and links posted on Astrobaki.

4.1.1. The `ugradio` *Python Package*

Modules are a way to distribute and reuse Python code. They are themselves just bundles code that you `import` into your program, where you can access everything as if you'd written it yourself. **Packages** are bundles of modules that can all be installed together. For this class, we will be using the `ugradio` package to provide supporting code for your labs. This package is already installed on the lab computers (you can test this by opening `ipython` and typing `import ugradio`). If you ever need to install it on another computer, all of the code is linked to from the course website.

You will make use of two modules inside of the `ugradio` package: `pico`, and `dft`. Take a minute to browse the code in these two modules so that you understand how to use it.

1. `ugradio.pico` — runs the A/D board to digitally sample signals. This is essential.
2. `ugradio.dft` — provides code for doing arbitrary Discrete Fourier Transforms. This contrasts the functionality of `numpy.fft`, which can only do certain kinds of Discrete Fourier Transforms, but can do them very fast.

4.2. Your First Digital Sampling: the Nyquist Criterion

We begin this course by exploring the all-important realms of the Nyquist criterion and aliasing in digital sampling. If you sample a signal too slowly, the signal won't be well-reproduced. But if you sample really fast, then you generate large data files that take a long time to process. Just how slowly can you sample the signal without completely losing its basic properties (such as, for example, the fact that it oscillates with frequency ν_{sig})?

The fundamental parameter here is the ratio of sampling frequency ν_{samp} to signal frequency ν_{sig}. With our equipment we can set ν_{samp} to only selected, quantized values (62.5 MHz/N, where N is a small integer). However, we can set ν_{sig} with almost arbitrarily high precision. So to explore these issues we will pick a sampling frequency ν_{samp} and take data at several signal frequencies ν_{sig}. Be sure to use a coax T so that you can look at the sampled signal on the oscilloscope. Set the peak-to-peak voltage appropriately so that it doesn't saturate the Analog-to-Digital Converter (known as the ADC).

Use `ugradio.pico` to get your data, which will appear in the form of a numpy array. In order to avoid having to re-take your data every time you run your program, you should save this numpy array to a file with either `numpy.save` or `numpy.savez`. Thereafter, you can just load your data from this file with `numpy.load`.

We want to explore sampling rate issues, so to that end we will begin by...

1. Pick a convenient sampling frequency ν_{samp}.
2. Set the synthesizer to frequency $\nu_{sig} = (0.1, 0.2, 0.3, \ldots, 0.9)\nu_{samp}$ and take data.

The sampler gives you `nsamples=16000` samples, which is to say that you can change how many samples you request, but it defaults to 16000. For this part of the lab, it's easier to deal with fewer, so just use the first N (in Python, with the command `firstN = samples[:N]`), with N being a few hundred or thousand. Throughout the datataking, you should always be monitoring the signal with the oscilloscope. These are sine waves, so it's easy to measure the period by looking at the oscilloscope; each time you digitally sample the signal, you should write down the period (maybe in your *lab notebook?*).

For each dataset, use Python to plot (with the `pylab`/`matplolib.pyplot` package) the digitally sampled waveform versus time. Make the plots informative, meaning that you label the axes (in particular, label the x-axis in time units) and you can clearly see the signal shape; if necessary, plot only a part of the data so you can clearly see the signal shape (e.g., a few cycles of the sine wave). Plot both the sampled points and the lines connecting the points (e.g., `pylab.plot(times, data, 'k.-')`). Compare your plot with the oscilloscope trace.

Also, for all the datasets derive and plot the Fourier power spectrum (see §5.2 and §5.3). Make sure that you *label the axes* with proper values of time and frequency—and choose convenient units, such as microsec (μs) and MegaHz (MHz), to avoid huge and tiny numbers. In deriving the Fourier spectra, use our homegrown DFT procedure (see §5.2).

Now, *look at both sets of these plots* and note any funny business. Think about your results and draw your own conclusion: just what is the minimum sampling rate that you can get away with? (That's **Nyquist's criterion)**.

4.3. Fourier Voltage and Power Spectra

Above we looked at the power spectrum and didn't examine the voltage spectrum. Now let's look at the voltage spectrum. The voltage spectrum numbers are complex, with real and imaginary parts. Plot the real and imaginary parts separately. It is most informative to have them on the same panel, which you can do either by plotting the real part and then overplotting the imaginary part in a different color; or you can use two plots one below the other by setting `pylab.subplot(2,1,1)` and `pylab.plot(freqs, spec.real)`, and then

`pylab.subplot(2,1,2)` and `pylab.plot(freqs, spec.imag)`. Take a detailed look at the plotted points with the goal of seeing if they exhibit any symmetry for negative and positive frequencies about 0. What do you see?

To make sure that any conclusions you draw regarding the symmetry are not just a fluke, repeat this process for several independent data streams. The mathematicians have a name for the kind of symmetry exhibited in these voltage spectra. It's called **Hermitian** symmetry.

What does it mean that the voltage spectra are complex? What do the real and imaginary parts represent? Is the imaginary part any less 'real' than the real part? Is it just a figment of your imagination? What does it mean that we talk of frequencies as being negative and positive? Are negative frequencies any less real than positive ones?

When you compare the plots for several independent data streams, do the voltage spectra repeat identically? Why not? What is happening when sometimes the real portions are positive or negative? When the imaginary portions have more amplitude than the real ones?

For the power spectra, repeat this symmetry examination and the test for repeatability. What kind of symmetry do the power spectral points exhibit? Apply to the power spectra the questions we posed just above for the voltage spectra.

Choose one of the power spectra and take its inverse Fourier transform. For this to work, you need to make sure your DFT correctly infers which array bins in your power spectrum correspond to which frequencies. Separately, calculate the autocorrelation function (ACF) directly from the voltage time series, either manually or with `numpy.correlate`. According to the correlation theorem, the FT of the power spectrum should equal the ACF. Does it?

4.4. Leakage Power

Above, you calculated a power spectrum for each input signal at N distinct frequencies separated by $\Delta\nu = \nu_{samp}/N$. In each, you found a spike corresponding to the input signal's frequency. Here, focus on just one of the properly-sampled signals ν_{sig}. Calculate the power spectrum for many more than N output frequencies than recommended in §5.2, i.e. make the frequency increment much smaller than $\Delta\nu = \nu_{samp}/N$. Making the output frequencies closer together gives a more nearly continuous frequency coverage in the plot of the output spectrum. Turn up the vertical scale a lot to see if there is any nonzero power at frequencies other than ν_{sig}. You *do* see such power! This is **spectral leakage**. It affects all power spectra calculated using Fourier techniques.

Can you understand what's going on from a mathematical viewpoint?

4.5. Frequency Resolution

If you had two sharp spectral lines, how closely spaced in frequency could they be and still resolve them? Investigate this experimentally by combining two SRS outputs in a power splitter, with the two SRS frequencies very close together, and plot the power spectrum. For this, you'll again want to plot points much more closely spaced in frequency than the $\Delta\nu = \nu_{samp}/N$ recommended in §5.2.

How close together can the two frequencies be for you to still be able distinguish them? This is called the **frequency resolution.** How does it depend on the number of samples you use in the DFT? In particular, how does it compare to the length of the time interval that those samples cover?

Can you understand this from a mathematical viewpoint?

4.6. Nyquist Windows

Above, we calculated Fourier spectra for frequencies in the range $\pm\nu_{samp}/2$. What do we get when we increase this range? Explore by taking a Nyquist-sampled time series and calculating the Fourier spectrum for a much larger frequency range, $\pm N\nu_{samp}/2$, where N is at least 4, retaining the original frequency interval. Each value of N gives you a spectrum in a different **Nyquist window**. How do the spectra in different Nyquist windows compare? Note that, for $N > 1$, you are calculating power spectra for frequencies that violate the Nyquist criterion. Nevertheless, the results aren't gibberish. In fact, in Lab 4 of the course we use a digital spectrometer that samples the 12^{th} Nyquist window.

This shows that the strictly correct statement of the Nyquist criterion is that the bandwidth—i.e., the frequency range of the signal—must not exceed ν_{samp}. For the first Nyquist window this is equivalent to the simpler statement of the Nyquist criterion that we explored at first.

4.7. FTs of Noise

A blackbody radiator with temperature T emits electromagnetic waves with specific intensity (power per area per Hz per solid angle) given by the blackbody formula $I =$

$2h\nu^3/c^2(e^{h\nu/kT} - 1)$. We are radio astronomers, which means we operate in the regime $h\nu/kT \ll 1$, so the blackbody formula goes to the much simpler Rayleigh-Jeans (RJ) limit, $I = 2kT/\lambda^2$. The noise power depends linearly on T. For a number of good reasons, radio astronomers choose to measure noise power in units of temperature and define a brightness temperature T_B such that $I \equiv 2kT_B/\lambda^2$ for an observed specific intensity.

When we observe, the electric field of the blackbody radiation is converted to voltage when it strikes the probe in the 'feed' of the telescope. The electric field and its corresponding voltage have the same statistical properties: Gaussian randomness and zero mean. Because of the randomness, it's called 'noise'. or more properly 'Gaussian random noise'. In the lab we have laboratory sources of noise. Explore the properties of digitally sampled noise:

1. Connect our noise generator to our $\sim$ 6-MHz wide bandpass filter (the Minicircuits SBP-21.4 filter) and take a 16000-point time series with the picosampler. These samples are voltages. What's the mean voltage (sum the voltages, divide by the number in the sum)? What's the mean square voltage (sum the squares of the voltages, divide by the number in the sum). What's the root-mean-square (rms) voltage (it's the square root of the mean square voltage)?

2. Plot a histogram of the sampled voltages (see Python's `numpy.histogram` function—for documentation, type `numpy.histogram?` in IPython). The histogram should look Gaussian, with a width equal to the rms voltage. Overplot this theoretically-expected Gaussian—does it look like your data?

3. Take a total of 32 16000-point samples with the picosampler (let's call these 32 'blocks' and derive the power spectrum for each block using the direct FT method. Plot the average of all 32 power spectra. What does this look like?

4. Plot the power spectrum for a single block and compare to the above average. Do the same for the average of N blocks, where $N = (2, 4, 8, 16)$ What you are doing here is looking at how integration time affects the signal-to-noise ratio (SNR): the 'signal' is what you see with long integration times and the 'noise' is the 'grass'. How does SNR depend on N? (Hint: SNR is proportional to N^x; what is x?)

5. Calculate the ACF for a single block, using the entire set of 16000 samples for delays of ≤ 2000 samples. Also derive the power spectrum from this ACF and compare with the direct FT-derived power spectrum for the same block. Are they identical? Compare the width (full width half max, or FWHM) of the ACF ($\Delta\tau_{FWHM}$) with the FWHM of the power spectrum (ΔF_{FWHM}). How do $\Delta\tau_{FWHM}$ and ΔF_{FWHM} compare?

5. IN THE MIND: FOURIER TRANSFORMS, THE ANALYTIC AND DISCRETE VERSIONS (First Week)

5.1. The Analytic Fourier Transform

A Fourier Transform (FT) maps a function between two complementary coordinates which, for now, are usually time, t, and spectral frequency, ν. The input to the forward Fourier Transform (FT) is a signal versus time, say $E(t)$; the output is a signal versus frequency, say $\tilde{E}(\nu)$. Mathematically, the Fourier Transform is the integral

$$\tilde{E}(\nu) = \int_{-T/2}^{T/2} E(t) e^{2\pi i \nu t} dt \ . \tag{1}$$

The input signal $E(t)$ is multiplied by the complex exponential and integrated, so the output $\tilde{E}(\nu)$ is complex. Of particular importance is that the Fourier Transform is **invertible**: you can go from the time to the frequency domain, and from the frequency domain you can get back to the time domain using the inverse transform

$$E(t) = \frac{1}{F} \int_{-F/2}^{F/2} \tilde{E}(\nu) e^{-2\pi i \nu t} d\nu \ . \tag{2}$$

This is because (complex) sine waves form a **basis** over the space of functions. Any function can be expressed exactly and uniquely by its Fourier coefficients. For those of you looking for applications of that linear algebra class you took, the Fourier Transform a linear matrix operation akin to a fancy rotation matrix. As such, it has lots of nice properties like: (1) **linearity** (the FT of a sum is the sum of the FTs), (2) **invertibility** (there is no information loss in the FT; it can be undone), and (3) unitarity (the FT is power-preserving). *Note:* If you're paying attention, you would wonder how the integration limits F and T are defined above. In the proper analytic formulation, they are both infinity. We emphasize their boundedness here because, in practice, i.e. when you do actual measurements or numerical calculations, neither can be infinity! Also note that, according to the Fourier conventions we've written here, our forward FT does not divide by the integration interval, but the inverse FT does. These conventions match `numpy`'s `fft` and `ifft` conventions.

5.2. The Discrete Fourier Transform (DFT)

Once we sample our signal at discrete interals versus time, it is no longer continuous. With the discrete transform, the integral becomes a sum. In this sum, you need to specify:

1. The set of sample times. I suggest:

(a) Using N samples, where N is even (and even better: a power of 2).

(b) Define the time range so that the center time is the zero point. With N even, there is no center time, so make the times run from $\frac{-N}{2}/\nu_{samp}$ to $(\frac{N}{2}-1)/\nu_{samp}$.

2. The output is a function of frequency, so you have to specify the frequencies for which you want the output $E(\nu)$. I suggest that, at first, you calculate the the output for N frequencies running from $-\frac{\nu_{samp}}{2}$ to $+\frac{\nu_{samp}}{2}\left(1-\frac{2}{N}\right)$. This makes the frequency increment equal to $\Delta\nu = \nu_{samp}/N$ over a total range of just under ν_{samp}. Thus, you calculate a **voltage spectrum** running from $-\frac{\nu_{samp}}{2}$ to not quite $\frac{\nu_{samp}}{2}$ using our in-house DFT procedure (see the `ugradio.dft` module). To find out how to use the DFT, you can type `ugradio.dft.<tab>` to see an auto-complete of what is available in the module. You can also type `ugradio.dft??` to see the code, and of course, you can type `ugradio.dft.dft?` to see the docs for the DFT function inside the `dft` module.

5.3. Power Spectra and Discrete Fourier Transforms

We are often interested in the output **power spectrum**, P_ν. Power is proportional to voltage squared. For complex quantities, the squaring operation means we want the sum of the squares of the real and imaginary parts. We obtain this by multiplying the voltage by its complex conjugate (denoted by '*'),

$$P_\nu = \tilde{E}(\nu)\tilde{E}^*(\nu)\ . \tag{3}$$

In Python, there are two ways to get this product. One is to use the `conj` function, i.e. `P = E * E.conj()`. Should the imaginary part of `P` be zero? (answer: yes! Why is this?) Is it? (answer: not always! Why not?) To get rid of this annoying and extraneous imaginary part, you can use the `float` function: `P = float(P)`. For arrays, this becomes `P = P.astype(numpy.float)`.

The other (more convenient and suggested) way is to square the length of the complex vector, i.e. `P = numpy.abs(E)**2`. The result is automatically real.

5.4. The Power Spectrum and the Autocorrelation Function (ACF)

There is a very important theorem involving Fourier transforms of two functions. It is called the **convolution theorem**. It has a cousin called the **correlation theorem**. Understanding these, and being able to apply them, is one of the requirements for being a real radio astronomer.

The convolution theorem: Consider two functions $E(t)$ and $F(t)$. They may be functions of either frequency or time; here, we take them as functions of time. The convolution of these two functions is

$$[E * F](\tau) = \int_{-T/2}^{+T/2} E(t)F(\tau - t)\ dt \tag{4}$$

and the correlation of the two functions is

$$[E \star F](\tau) = \int_{-T/2}^{+T/2} E(t)F(\tau + t)\ dt \tag{5}$$

Conceptually, these two functions describe sliding F over E by changing the parameter τ. τ is called the 'time delay', or simply the 'delay'. These two expressions are almost identical; the only difference is the sign of t in the argument of F. If F is symmetric, which is the case of interest for us, the two are identical. Using our definition of the Fourier transform from Equation 1, the convolution theorem states:

$$\widetilde{[E * F]}(\tau) \equiv \int_{-T/2}^{T/2} [E * F](\tau)\ e^{2\pi i \tau \nu} d\tau = \tilde{E}(\nu) \cdot \tilde{F}(\nu) \tag{6}$$

and the correlation theorem:

$$\widetilde{[E \star F]}(\tau) \equiv \int_{-T/2}^{T/2} [E \star F](\tau)\ e^{2\pi i \tau \nu} d\tau = \tilde{E}(\nu) \cdot \tilde{F}^*(\nu) \tag{7}$$

These theorems apply strictly only in the limit $T \to \infty$ (because of 'end effects' when T is finite), but for finite T—the case for any real measurement—their equality is 'good enough'. In words: **The FT of the convolution in the time domain is equal to the product of the Fourier transforms in the frequency domain.** Ditto for the correlation theorem, except that one of the FTs is complex-conjugated. If $F(t)$ is symmetric, then the imaginary part of its Fourier transform is zero, which means $\tilde{F}^*(\nu) = \tilde{F}^(\nu)$, and two theorems become identical.

A hugely important application of this theorem is the case when $E(t) = F(t)$, in which the correlation function becomes the *Autocorrelation function* $ACF(\tau)$, and equation 7 states, in words: **The power spectrum is equal to the Fourier transform of the ACF**

We'll talk about other examples and applications in class.

When calculating a digital version of the correlation function, you have to worry about 'end effects'. Suppose you are calculating an ACF for N samples with delays ΔN ranging up to $N/2$. Then the number of terms in the sum is always smaller than N because the delays

'spill over the edge' of the available samples. So when you calculate the ACF you need to properly normalize:

$$ACF(\Delta N) = \frac{\Sigma_{k=0}^{N-\Delta N-1} x_k x_{k+\Delta N}}{\Sigma_{k=0}^{N-\Delta N-1} x_k^2} \tag{8}$$

5.5. The Fast Fourier Transform (FFT)

Above in §5.2, you had N time samples and evaluated the DFT for N well-chosen frequencies. These were "well-chosen" because for these particular values of frequency—and *only* these particular values—you can get back to the time domain by using the inverse transform (in Python using `ugradio.dft.idft`).

It so happens that, for these particular combinations of frequency and time, there is a very fast algorithmic implementation called the *Fast Fourier Transform*, the FFT. What do we mean by "Fast"? Normally when you do a DFT, you have N input numbers and N output numbers and the number of calculations $\propto N^2$. When N gets large, this takes a long time! For the FFT, on the contrary, the number of calculations $\propto N\log(N)$, and this makes it possible to do large-N transforms.

Try Python's FFT (in the `numpy.fft` module) and compare it to your DFT calculation above. The FFT output is ordered in what you might think is a funny and awkward way: when Fourier transforming a time series to obtain N frequencies, the frequency array is ordered with $\frac{N}{2} - 1$ positive frequencies first, then $\frac{N}{2}$ negative frequencies However, it's really not awkward for most applications. See our "DFT's with DFT's" handout for details.

From now on, use FFT instead of DFT—unless you need results for additional output points, either more closely-spaced or over a broader range.

6. IN THE LAB: MIXERS (Second Week)

6.1. The Double-sideband Mixer (DSB Mixer)

Figure 1 shows a block diagram of a DSB mixer, whose backbone is the device called a mixer, which multiplies the two input signals. It's simple: the r.f. signal goes into one mixer port, the l.o. goes into the second mixer port, and the i.f. output is the third port.

For the mixer use a Mini-Circuits ZAD-1, which has three BNC connectors (three **ports**)

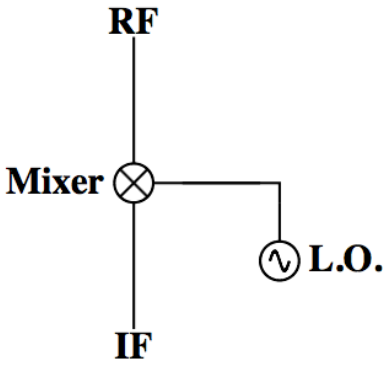

Fig. 1.— A DSB mixer. In the text, we sometimes refer to the r.f. input as the 'signal'.

and works well at these frequencies. The ZAD-1, like nearly all mixers, has its ports labeled "R" (the "RF" or "signal"); "L" (the "local oscillator"); and "X" (the "mixing product") or "I" (the "intermediate frequency"). The ZAD-1 is a balanced mixer, so the "R" and "L" ports are identical, and in particular will not couple to DC or very low frequencies. To find out the frequency ranges a ZAD-1 supports on each input, I encourage you to look up its datasheet online. In contrast, the "I" port is coupled differently and will handle voltages all the way down to, and including, DC. The mixing process functions no matter which two ports are used as inputs. For example, if you are using a mixer to modulate a high frequency (say, a few MHz) with a low frequency (say, a few kHz), you should use the "I" port for the low frequency and either of the other two for the high frequency; take the output from the third port.

For this, use two SRS synthesizer oscillators as inputs to a mixer to explore the spectra and waveforms in the DSB mixing process. One SRS synthesizer works up to 30 MHz (the fancy one), and the other works up to 15 MHz. Assign one of the SRS synthesizers to be your "local oscillator" (LO) with frequency ν_{lo}, and the other your "signal" with frequencies $\nu_{sig} = \nu_{lo} \pm \delta\nu$. Here, you choose the frequency difference $\delta\nu$ and you set the two synthesizers, one to the lo frequency and the other to the signal frequency. There are two cases for the signal frequency, $\nu_{sig} = \nu_{lo} + \delta\nu$ and $\nu_{sig} = \nu_{lo} - \delta\nu$. Make $\delta\nu$ somewhat small compared to ν_{lo}, maybe 5% of ν_{lo}. For the input power level, a good choice is 0 dBm[1] for both synthesizers. The output consists of both the sum and difference frequencies, so choose the ports appropriately.

[1]What does this "dBm" mean? It's the power relative to 1 milliwatt, expressed in decibels (dB). For our system the cable impedance is 50 ohms; what's the rms voltage for a signal with power level 0 dBm?

We will want to digitally sample the mixer output and explore both the sum and difference frequencies. As you learned above, there are extremely important issues regarding sampling rate. The most basic is the Nyquist criterion. Here, we also want enough samples per period to give you a reasonable visual facsimile of the sine wave when you plot it; from this standpoint, it's nicer to sample at twice Nyquist, or even faster. Another issue is the number of points you sample, which must be large enough to give you at least a few periods of the slowest sine wave.

For the two cases $\nu_{sig} = \nu_{lo} \pm \delta\nu$, plot the power spectra versus frequency. Explain why the plots look the way they do. In your explanation include the terms "upper sideband" and "lower sideband".

For one of the cases, plot the waveform. Does it look like the oscilloscope trace? Also, take the Fourier transform (not the power spectrum) of the waveform and remove the sum frequency component by zeroing both the real and imaginary portions (this is 'Fourier filtering'). Recreate the signal from the filtered transform by taking the inverse transform and plot the filtered signal versus time. Explain what you see.

6.2. Real Mixers: Intermodulation Products

Look at one of the above power spectrum plots with the gain turned up so you can see weak signals. What do you see? A forest of lines! What are these?

We describe a mixer as an ideal device that multiplies the two input signals. However, real mixers are not ideal. They function by using nonlinear diodes to perform an approximate multiplication. A real mixer also produces harmonics of the mixed input signals. And it produces the product of harmonics of each input signal times the other, vice-versa, and even harmonics of each input signal with itself—in essence, whatever signal is present inside the mixer will be combined with every other signal. These undesired products produce nonideal signals, which are **intermodulation products**; engineers fondly call them 'intermods' or, more colloquially, 'birdies'. When a well-designed mixer is operated with the proper input signal levels, the intermods have much less power than the main product, but they can nevertheless ruin sensitive measurements.

Look at your forest of lines and see if you can identify how some of the stronger ones come about.

6.3. The Sideband-Separating Mixer (SSB Mixer)

Figure 2 shows a block diagram of a SSB mixer. It's only a little more complicated than the DSB mixer: it consists of two identical DSB mixers, one on the left and one on the right, fed by the same l.o. *Note the important part:* the right-hand l.o. is delayed by 90° relative to the left-hand one, which means that the mixing product on the right is delayed by 90° with respect to the left. This means we can regard the right-hand output as the real part and the left as the imaginary part of a **complex vector.** We sample both outputs simultaneously and use them as the complex input to the Fourier transform; the resulting power spectrum shows both negative and positive frequencies. Engineers and geeks call this 'IQ sampling'.

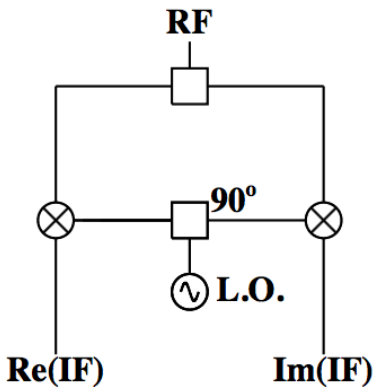

Fig. 2.— An SSB mixer. The important part is the 90° phase delay in the right-hand l.o. This is normally achieved with device called a 'quadrature hybrid'. We will achieve it with a $\lambda/4$ piece of cable.

From the block diagram in Figure 2, construct an SSB mixer that achieves the phase delay with a cable[2]. We will use it to experiment with no phase delay (a short cable) and a 90-degree phase delay (a long cable). For experimentation with this two-output mixer, use the two SRS synthesizer oscillators as inputs, as before.

6.3.1. As a DSB Mixer

First see what happens when the phase delay cable is short (ideally zero), so that the two halves are essentially identical and have only a small relative phase delay. Pick a value for $|\delta f|$ and take time series data for the two corresponding values of f_{sig} (these are the upper and lower sidebands). Calculate the power spectra. When taking the Fourier transform, be

[2]Somewhere around the lab we have labelled a cable as being $\lambda/4$ at 21 MHz.

sure to make the inputs complex—you have two simultaneous samples, one real and one imaginary. Looking at the power spectra alone, can you distinguish between positive and negative δf?

6.3.2. The SSB Mixer

Now see what happens when the phase delay cable introduces a relative phase delay of 90° between the l.o. signals going to the two mixers. Repeat what you did above in §6.3.1. Looking at the power spectra alone, can you distinguish between positive and negative δf?

If you have the time and inclination, verify that the phase difference between the two mixer outputs behaves as shown in Figure 3. Why does it behave this way?

7. IN THE MIND: ON MIXERS AND THE HETERODYNE PROCESS

7.1. Some Commentary: The Heterodyne Process

Mixers are important because they allow us to shift the frequency of the whole input spectrum by a uniform amount, ν_0. They do this by multiplying the input signal by a sine-wave **local oscillator** (LO) with frequency ν_0 (though we will use angular frequency, ω_0, below for cleaner notation).

In radio reception, this is vital because our *detectors* usually work best over a fixed frequency range, but our signals come in at many different frequencies. For example, for an AM station playing rock music, the ultimate detector is our ear, which works only at audio (kHz) frequencies; however, the AM stations transmit at much higher frequency, nearly 1 MHz. A mixer is used to shift the frequencies of the AM station down to the audio region. Such receivers are called **heterodyne** receivers, and this principle is used universally not only in consumer radios, TV's, and cellphones, but also more nobel pursuits, like radio astronomy.

7.2. Some Theory: The Single Sideband (SSB) Mixer

Even though we do the SSB Mixer second in the lab, the theory of a SSB is probably more straightforward to understand, so long as we are willing to use complex numbers and allow for the existence of **negative frequencies**. Negative frequencies might seem a little

weird; how can something oscillate a negative number of times per second?

To understand, let's use Euler's formula to write a complex sinusoid as

$$Ae^{iwt} = A\cos(\omega t) + i \cdot A\sin(\omega t). \tag{9}$$

In some ways, this complex sinusoid is the "true" sine wave. The real-valued versions are built out of pairs of complex sine waves:

$$\cos(\omega t) = \frac{1}{2}(e^{i\omega t} + e^{-i\omega t}) \tag{10}$$

$$\sin(\omega t) = \frac{1}{2i}(e^{i\omega t} - e^{-i\omega t}) \tag{11}$$

Now, once we've defined a complex sine wave (which very soon will just be called a "sine wave"), we can switch $-\omega$ for ω,

$$Ae^{i(-w)t} = A\cos(\omega t) - i \cdot A\sin(\omega t), \tag{12}$$

which is to say that the negation of a frequency swaps the sign of the imaginary component. So rather than thinking of a negative frequency as "negative Hertz", let's instead think of it as a phase relationship between the sine and cosine components. In fact, if we take $x = \cos\omega t$ and $y = \sin\omega t$, it's a phase relationship that, for $\omega < 0$, makes (x, y) run the opposite direction around a circle from $\omega > 0$.

Now let's take an idealized SSB mixer that has a LO that is a complex sinusoid $e^{-i\omega_0 t}$ of unity amplitude with a negative frequency. This LO is mixed (multiplied) by an input signal. As an example, let's take an input signal that is the sum of two real-valued sine waves,

$$E(t) = A\sin(\omega_0 - \Delta\omega)t + B\sin(\omega_0 + \Delta\omega)t. \tag{13}$$

We can then use Euler's formula to express the product output by the mixer as

$$E(t) \cdot e^{-i\omega_0 t} = A\sin(\omega_0 - \Delta\omega)t \cdot e^{-i\omega_0 t} + B\sin(\omega_0 + \Delta\omega)t \cdot e^{-i\omega_0 t} \tag{14}$$

$$= \frac{A}{2i}\left[e^{i(\omega_0-\Delta\omega)t} - e^{-i(\omega_0-\Delta\omega)t}\right]e^{-i\omega_0 t} + \frac{B}{2i}\left[e^{i(\omega_0+\Delta\omega)t} - e^{-i(\omega_0+\Delta\omega)t}\right]e^{-i\omega_0 t} \tag{15}$$

$$= \frac{A}{2i}\left[e^{-i\Delta\omega t} - e^{-i(2\omega_0-\Delta\omega)t}\right] + \frac{B}{2i}\left[e^{i\Delta\omega t} - e^{-i(2\omega_0+\Delta\omega)t}\right]. \tag{16}$$

After mixing, each component sine wave in $E(t)$ has a term that appears at a beat frequency $e^{\pm i\Delta\omega t}$, as well as a component that appears a much higher frequency near $2\omega_0$. These higher-frequency components are typically filtered off using a low-pass filter (LPF), leaving just the beat-frequency terms

$$\text{LPF}\left[E(t) \cdot e^{-i\omega_0 t}\right] = \frac{A}{2i}e^{-i\Delta\omega t} + \frac{B}{2i}e^{i\Delta\omega t}. \tag{17}$$

These terms retain amplitude and $\Delta\omega$ of their original signal, but have been shifted down to a lower frequency where they can be more easily sampled and processed[3].

Furthermore, so long as we retain both the real and imaginary components (which, in reality, are just the components of the original signal that were multiplied by the cosine and sine components of the LO, respectively), we can distinguish between the A and B signal components by whether they appear at positive or negative frequency. This ability to distinguish between positive and negative **sidebands** is why we call this a Single-SideBand Mixer: we can look at each sideband separately. If we didn't have both the cosine and sine components, we would not be able to distinguish positive and negative frequencies. Signals A and B would then sit on top of each other, and we would have no choice but to look at both sidebands simultaneously.

Figure 2 shows a block diagram of the SSB mixer. The RF input and the LO are each split by a power splitter so that we have two identical mixers, one on the left and one on the right, whose outputs are labelled **Re(IF)** and **Im(IF)**, respectively. The one on the left is identical to the DSB mixer in figure 1. The one on the right differs in only one way, which is crucial: its LO is delayed by 90° relative to that on the left. With this, the **Im(IF)** output lags the **Re(IF)**, becoming a sine wave to the **Re(IF)**'s cosine.

7.3. Some Theory: The Double Sideband (DSB) Mixer

We now turn to the theory of the DSB mixer, which is very straightforward to build (the LO is just a real-valued sine wave, as is the RF, so we only require one mixing circuit), but a bit more complicated to understand and get to work.

In this case, we are going to repeat the SSB derivation, but instead of using $e^{-i\omega_0 t}$ as our LO, we will use $\sin\omega_0 t$. To begin, let's take our signal to be $E(t) = A\sin(\omega_0 + \Delta\omega)t$. In

[3]In real life, e.g. a radio station, the "signal" is speech or music which spans a range of δf. In astronomical life, e.g. the 21-cm line, the "signal" is a Doppler broadened line, which again has a broad range of δf. In both cases, a mixer with a well-chosen LO frequency can be used to "mix" the signal down to low frequency, where it can be sent to a speaker (if you are listening to the radio, or if you are Jodie Foster's character *Ellie* in *Contact*.

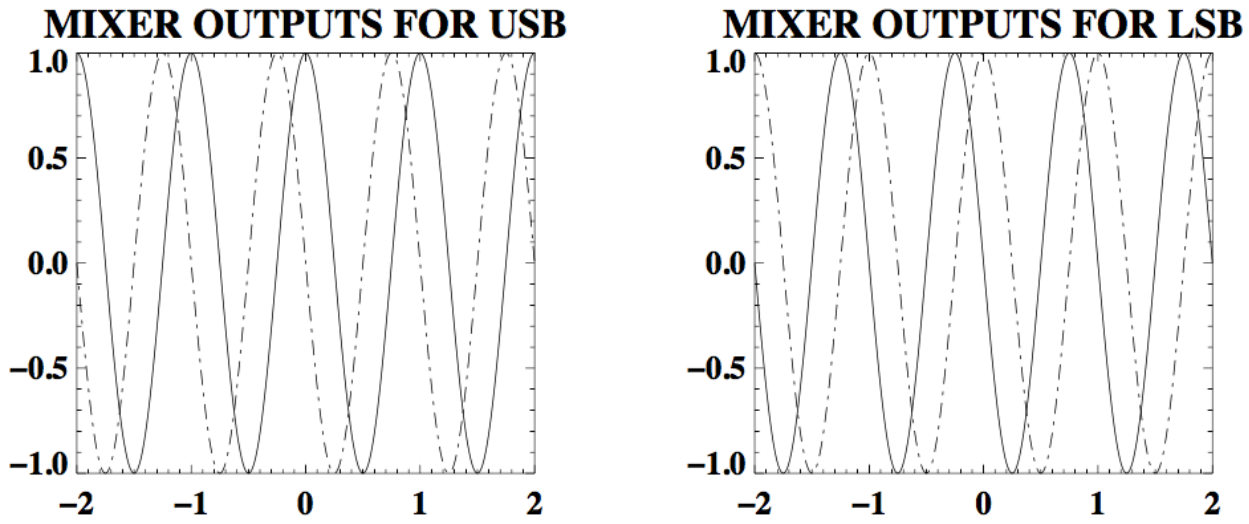

Fig. 3.— Outputs of the first mixers for the two sideband cases. Dashed curve shows the left-hand mixer, solid is the right-hand mixer. Left panel shows $\delta\omega > 0$ (upper sideband—USB); right panel shows $\delta\omega < 0$ (lower sideband—LSB).

this case, the output of our mixer becomes

$$E(t) \cdot \sin\omega_0 t = A\sin(\omega_0 + \Delta\omega)t \cdot \sin(\omega_0 t) \tag{18}$$

$$= \frac{A}{2i}\left[e^{i(\omega_0+\Delta\omega)t} - e^{-i(\omega_0+\Delta\omega)t}\right] \cdot \frac{1}{2i}\left[e^{i\omega_0 t} - e^{-i\omega_0 t}\right] \tag{19}$$

$$= \frac{A}{2}\left[e^{i\Delta\omega t} + e^{-i\Delta\omega_0 t} - e^{i(2\omega_0+\Delta\omega)t} - e^{-i(2\omega_0+\Delta\omega)t}\right] \tag{20}$$

$$= A\left[\cos\Delta\omega t - \cos(2\omega_0 + \Delta)t\right]. \tag{21}$$

As in the SSB, the mixer output has two components: a beat frequency (which is our desired output) and a component near $2\omega_0$ (which we typically remove using a LPF).

The unfortunate part about the DSB mixer becomes obvious if we consider the case we used for the SSB mixer, where $E(t) = A\sin(\omega_0 + \Delta\omega)t + B\sin(\omega_0 - \Delta\omega)t$. Repeating our algebra above for each component above, we find that

$$E(t) \cdot \sin\omega_0 t = A\left[\cos\Delta\omega t - \cos(2\omega_0 + \Delta)t\right] + B\left[\cos(-\Delta\omega)t - \cos(2\omega_0 - \Delta)t\right]. \tag{22}$$

But $\cos(-\Delta\omega)t = \cos\Delta\omega t$, so after removing the high-frequency ($2\omega_0$) components with a LPF, we end up with

$$\mathrm{LPF}\left[E(t) \cdot \sin\omega_0 t\right] = (A + B)\cos\Delta\omega t. \tag{23}$$

Which, you can see, has two signals that were at two different fequencies ($\omega_0 + \Delta\omega$ and $\omega_0 - \Delta\omega$) added on top of each other.

The DSB mixer, though simpler to build, cannot distinguish between postive and negative deviations around the LO frequency. It stacks them right on top of each other, so that any frequency you look at in the output can be the sum of two different signals. This is what lends it the name "Double Sideband".

Three things are important here:

1. The two sidebands—the two different input frequencies ($[\omega_{s-} = \omega_0 - \delta\omega]$ and $[\omega_{s+} = \omega_0 + \delta\omega]$)— produce the same symmetric-around-zero pair of IF output frequencies $\pm|\delta\omega|$. The DSB mixer cannot distinguish between the two input frequencies.

2. Consider how $|\delta\omega|$ depends on ω_0: for the upper sideband, $\frac{d|\delta\omega|}{d\omega_0} = -1$, while for the lower $d\frac{|\delta\omega|}{d\omega_0} = +1$. We hope that the upper three panels of Figure 4 elucidate the situation.

3. A value of E_s for one sideband produces a certain mixer output power; the same value of E_s for the other sideband produces the same power. With regard to power, the sidebands are *indistinguishable*.

Figure 4 illustrates these results. The top panel shows the original RF spectrum, which consists of signals above the LO (the USB signal) and below (the LSB). Suppose you use a bandpass filter to eliminate the LSB. Then you have only the USB, and the second panel shows the IF spectrum after DSB mixing: the USB appears at both negative and positive frequencies and the spectrum is symmetric, meaning that the negative frequencies give exactly the same result as the positive ones.

Now use a bandpass filter to eliminate the USB, leaving only the LSB; the third panel shows the resulting IF spectrum.

If you didn't use any bandpass filters, then both the LSB and the USB would appear in the IF spectrum, as in the fourth panel. Looks complicated! With a DSB mixer, you can't distinguish between LSB and USB. The LSB and USB are inextricably mixed and you get the sum of the power spectra. The only way can achieve the rejection of either the LSB or the USB is by using an appropriate **bandpass filter** on the input RF spectrum.

But, nirvana! The bottom panel shows that SSB (Sideband Separating, or Single Sideband) mixing retains the sideband separation and identity.

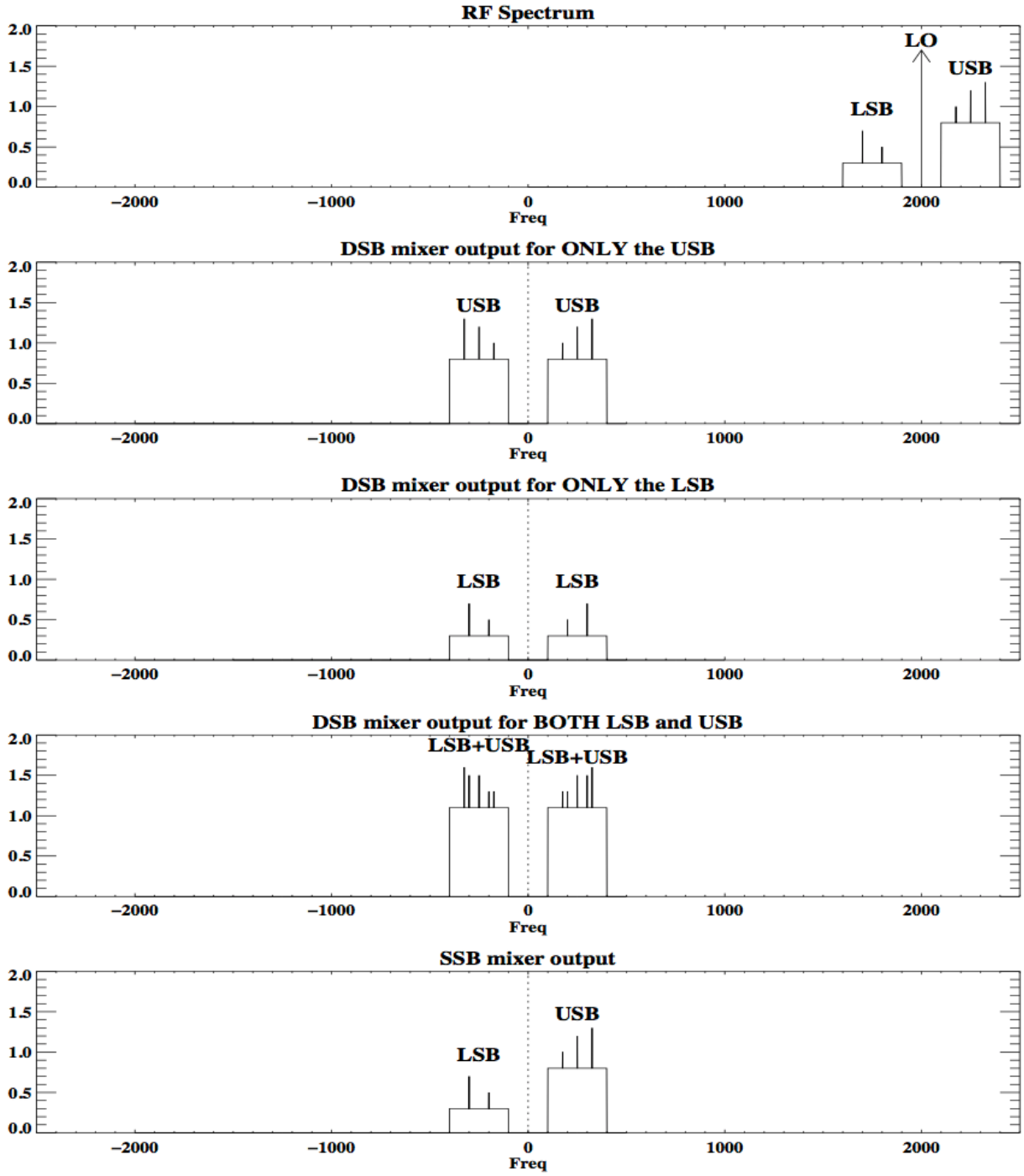

Fig. 4.— Upper and lower sidebands in DSB and SSB mixers for a set of δ-function test signals on top of broad level noise spectra. Top: the RF spectrum. The next two show the USB and LSB individually when they undergo the DSB mixing process; panel 4 shows how they both add together. The bottom panel shows the SSB mixer, which keeps them separate.

8. ON PAPER: YOUR LAB REPORT (Third Week)

8.1. Handouts

1. What should your lab report look like? `labreport_comments.pdf` *"SUGGESTIONS FOR LAB REPORTS"*

2. You must use LaTeXfor your lab report! `sample.pdf` "LaTex Is Your Friend OR ENEMY?" Answer to this question is a resounding YES for 'Friend'—if you have followed his handout. Use LaTeXfor preparing your lab report!

3. Now's the time for another look at efficient use of your text editor, because if you learn the keystoke commands you'll be much quicker and save lots of time further down the road.

4. You'll need to show plots into your lab report. To do this you make PDF files of your plots.