LAB 1 – January 21, 2020
EXPLORING DIGITAL SAMPLING, FOURIER TRANSFORMS, and both DSB and SSB MIXERS

Contents

1. Introduction

In this lab, we experimentally investigate digital samplers, discrete Fourier Transforms, and mixers. These are tools of the trade in radio astronomy, and you use them every day when you use a cell phone, listen to the radio, or watch television. Did you know that wifi was invented by a radio astronomer?

To get firm footing in the methods of radio astronomy, you will perform a series of experiments that illustrate the utility and hazards of these various techniques. You will configure equipment in the lab, acquire and analyze data, and store and illustrate the results. The signal processing pipeline you build in this lab will be re-used in Lab 2 to observe 21-cm line emission from hydrogen in the Milky Way, so put in the time to make it work and understand it. The effort you invest now will pay dividends later in the class.

This first lab is an excellent time to establish good research and laboratory practices. Remember that your grade is determined by the report you turn in three weeks from now. All of the exercises that fallow are aimed at building your understanding and giving you the raw materials (and data) out of which to build that report. You don't have to finish

everything, or proceed in the in order presented, but you do have to write a quality report that addresses the goals in §2.

Here are some specific suggestions for turning these lab instructions into a successful report:

- Keep careful notes, preferably in a **lab notebook** that you write in as you hook things up. Good notes document how the data are derived from the experimental setup so you can describe it accurately and track down mistakes. If you show your knowledgable GSI a head-scratcher of a plot but cannot remember which port of the mixer you plugged the noise into, they may have trouble diagnosing your problem.
- Back up (or revision control) your data, code, and plots. Nothing is more demoralizing than re-doing work or being unable to find the combination of parameters that reproduce a result.
- Start your report early. If you plan for which equipment, data, and plots you need for your report, your time in the lab will be spent more efficiently and you will avoid the panic that comes from writing up at the last minute.
- Formal lectures, the pages linked on AstroBaki, and these lab instructions are resources to help you write a strong report. As in real-life research, there is no one text book containing everything you need to know. Explore, take notes, and show me what you learned in your report. Remember, I can only grade you on what you write down. Make sure what you write down is your own work, in your own words. Cite sources external to this class (no need to cite AstroBaki, lecture, or the lab instructions).
- Your reports will be in the style of scientific publications. They are narratives of explanation and discovery that bring the reader—a science colleague—to the point of being able to understand and have confidence in your results. You may be presenting your work, but this report is about the *reader's* journey, not yours. It is a presentation of synthesized results with the background and salient details necessary to understand them. It is *not* a play-by-play of everything you did, all the plots you made, and all the wrong turns along the way.

2. Goals

The goals of this lab—the results your report should demonstrate you have achieved—are as follows.

- Learn how to sample electronic signals and convert them into digital signals. Demonstrate the phenomenon of aliasing and quantitatively relate that to the Nyquist criterion.
- Learn how to correctly use and display Discrete Fourier Transforms (DFTs) to determine the frequency power spectrum of a time series. Correctly calculate and label frequency and power axes in plots and demonstrate understanding of how frequency ranges and resolutions are determined from the duration and cadence of samples in a time series.
- Learn about negative frequencies. Motivate and measure how complex inputs to a Fourier Transform break the positive/negative frequency degeneracy.
- Learn about the origin of noise in electronic measurements and explore how the noise behaves under the Fourier Transform.
- Learn about the convolution/correlation theorem, demonstrate how spectral leakage in a power spectrum can be understood in this framework, and identify the relationship between autocorrelation functions and power spectra.
- Learn the basics of mixing for frequency conversion (the heterodyne technique) and explore how electronic mixers differ from ideal ones. Construct double- and single-sideband mixers and use measurements to demonstrate the theoretical and practical differences in their operation.

In order to achieve these goals, you will be developing technical skills that are crucial, but need not be written about explicitly in your report. (The quality of the report itself will demonstrate your growing proficiency.) These include:

- Using the Python programming language for analysis, signal processing, and plotting,
- Learning enough LaTeXto write up your results in a formal lab report that including an abstract, captioned figures, tables, and citations.
- Gaining familiarity with configuring laboratory equipment and navigating the computing environment.

3. Schedule

This is an ambitious lab with a steep learning curve. It is important not to get behind or backload your work schedule. In particular, writing the report takes a lot longer than you might think.

1. *Week 1.* Finish §5 and read the accompanying material in §6. *Be prepared to show work, software, and results to the class.*
2. *Week 2.* Finish §7 and read §8. Again, be prepared to present to the class.
3. *Week 3.* Read the handouts in §9 and write your formal report. Refer to our handouts and linked material for tips on how to structure a scientific paper. You will be graded on the degree to which your report addresses the specific goals in §2 and conforms to standards of a quality scientific publication.

4. Software Engineering

The programming required to complete these labs will increase in scale throughout the semester. As it does, 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 it may be one of the most marketable skills you learn in this class.

Learning to engineer good software can take a lifetime, but let us start with a few principles in this lab.

- *Package your code.* Keep files containing code together in directories that are separate from the data you acquire. Organize your code into functions and classes, and put the definitions in separate, importable files called modules. Modules separate the definition of the code from the scripts and notebooks you use to execute code with changeable parameters. Packaging code vastly improves organization, testability, and the degree to which you can re-use good code. Functions, classes, modules, and installable **packages** are easy to implement in Python, and there are many resources online to teach you how to use them.
- *Code (at least) twice.* It is nearly impossible to write well-organized code while figuring something out for the first time. You need room to hack at the problem, try and fail, learn and iterate. Code once to get a sense of how your program should work, then re-code it to get the organization and interfaces right. Time spent organizing and testing code up front will pay dividends later.

- *Test your code.* How do you know your code is doing what it should? Yes, it may run without raising an exception, but that doesn't mean it did what you intended. In software as in science, you can only trust something to the extent that you have tested it. Keep your tests organized so that you know what you have tested, and how. Modular testing (a.k.a **unit testing**) is the key to success in large software projects, particularly ones involving multiple people. The `unittest` module is commonly used in Python, but other modules have their own merits.

- *Control your revisions.* **Revision control** uses a specialized program (`git` is popular) to track the changes made to your code. They allow you to edit code with impunity, knowing that you can always ask for a report on what you changed and where, and you can always backtrack to a previously committed state of the code if you screw something up. I suggest opening a (free) GitHub account, initializing a project for this class, and then committing to it regularly. GitHub works with `git` to back up your work remotely, and it also allows you to easily synchronize your work across multiple computers. It is seriously worth the learning curve.

Advanced software engineering combines all of the steps outlined above, using online repositories to hold various branches of a software project, and when any of a community of coders push their changes, a continuous integration system runs unit tests on the code to make sure that nothing was broken, that the new codes defines new tests it must pass, and that all the code conforms to reasonable documentation and legibility standards. Code reviews examine changes that are made and submitted through pull requests, and then the changes (often multiple in parallel) are merged into the master code base of the project.

We don't need the advanced machinery for this class, but it is good to be aware of how coding communities work and to start down the path of good coding practices yourself. Try to improve your software engineering practices using the suggestions above.

At a minimum, firmly separate data acquisition (the code that runs and interacts with equipment in the lab and at the telescope) from data analysis. For the former, scripts that run from the command line are most appropriate. For the latter, `jupyter` notebooks are a fantastic tool. If you have code that needs to run under both environments, write modules that you can import into either.

5. Digitally Sampling a Sine Wave (Lab Activity, Week 1)

5.1. Handouts and Software

Your work this first week will require you to navigate the Linux operating system, using your (Vi/Emacs/3rd party) text editor to write Python. On Astrobaki[1] we have linked primers on Linux/Unix, Python Installation and Basic Programming, Unix Text Editors, and Revision Control. You can also review the key course content for this week: Nyquist Sampling and the Fourier Transform.

5.1.1. 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 on GitHub and 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.

- `ugradio.pico` — runs the PicoSampler Analog-to-Digital Converter to digitally sample signals. This is essential.
- `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.

5.2. Your First Digital Sampling: the Nyquist Criterion

In class, we learned about the Nyquist criterion for sampling discretely in time. Sampling too slowly results in a signal being aliased, but oversampling generates excessively large data files. Let us determine the optimal balance by sampling a sine wave and comparing

[1] https://casper.ssl.berkeley.edu/astrobaki/index.php/Undergraduate_Radio_Lab

the recovered signal to the original for different ratios of the signal frequency (ν_0) and the sampling frequency (ν_s).

In the lab, we will sample signals using a PicoSampler 2000 Analog-to-Digital Converter (ADC). The sampling frequency is set during data acquisition with the `capture_data` function in `ugradio.pico` module, but the ADC only samples at quantized values of 62.5 MHz/N, where N (set by the `divisor` parameter) is a small integer. However, we can set ν_0 with arbitrarily high precision using a signal generator, so instead of trying different sample frequencies for a fixed ν_0, we will do the reverse:

- Pick a convenient sampling frequency ν_s, then
- Set the synthesizer to frequency $\nu_0 = (0.1, 0.2, 0.3, \ldots, 0.9)\ \nu_s$ and take data.

We suggest using a co-axial T joint so you can put a copy of the signal you are sampling into an oscilloscope. Good science is about inspection, verification, and documentation. In that spirit, inspect the signal you are sampling at each step with the oscilloscope, verify that it matches that amplitude and period you set on the function generator, and document the settings used and files generated in your lab notebook.

Use `ugradio.pico.capture_data` to get your data, which will appear in the form of a numpy array. You should save this array to a file with `numpy.save` or `numpy.savez`. Your analysis scripts can then load data from this file with `numpy.load`. 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. Be sure to set the peak-to-peak voltage of the ADC so the input signal does not saturate it, but is much bigger than the quantized states of the ADC.

For each dataset, use the `matplotlib.pyplot` package (typically imported as `import matplotlib.pyplot as plt` to plot the digitally sampled waveform versus time. For plotting, it is easier to visualize fewer samples, so if you load a wavefrom from file into a variable `data`, you can slice off the first, say, 200 samples with the command `data = data[:200]`. Make the plot informative by putting the x axis in time units (e.g. `plt.plot(tiimes, data)` instead of `plt.plot(data)`). Does the period match what you expected?

If you would like to use this (or any) plot in your report, make sure you label both axes (with approprite units to avoid tiny or huge numbers) give it a title, and restrict the range to an appropriate scale where you can clearly see the signal shape and frequency. You will also need to come up with an appropriate caption that stands alone as a description of the plot and what it demonstrates.

For each of the datasets you take, derive and plot the Fourier power spectrum (e.g. the

square magnitude of the voltage spectrum; see §6.2 and §6.3). You can use `numpy.fft` to do the Fourier transform, but we suggest using our homegrown `ugradio.dft` version until you have mastered dealing with Fourier units and the subtleties of `fftshift`.

Now, examining the results, draw your own conclusion: what is the minimum sampling rate ($\nu_{s,min}$) that accurately reproduces the spectral frequency ν_0 in the sampled data? That is **Nyquist's criterion**.

5.3. Voltage Spectra and Power Spectra

Power spectra show which frequencies are occupied by signal power by telling us about the magnitudes of the complex numbers in the Fourier transform. Voltage spectra, by contrast, give us phase information, with real and imaginary parts.

What does it mean, that the voltage spectra are complex? What do the real and imaginary parts represent? Is the imaginary part less 'real' than the real part? What does it mean, for frequencies to be negative versus positive? These are questions your lab report could try to answer.

For one of your data captures in the previous section, plot the real and imaginary parts (e.g. `spec.real` and `spec.imag`) of the voltage spectrum on the same panel in different colors. Do the plotted points exhibit any symmetry between negative and positive frequencies?

Repeat this process on independent data streams to ensure the result is not a fluke. When you compare the plots for several independent data captures of the same sine wave, 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? Why might we use power spectra instead of voltage spectra, and vice versa?

Choose a power spectrum and take its inverse Fourier transform. For this to work, you need to make sure `dft.idft` correctly infers the frequencies corresponding to each bins in your power spectrum array, Separately, calculate the autocorrelation function (ACF) directly from the voltage time series manually with `dft/idft`, with `numpy.correlate`, and with `scipy.signal.correlate`. According to the correlation theorem, the Fourier transform of the power spectrum should equal the ACF. Does it? Explain any differences.

5.4. Leakage Power

By default, `dft` calculates a power spectrum at N frequencies for a signal with N time samples, each frequency sepearated by $\Delta\nu = \nu_s/N$. (For Fast Fourier Transform (FFT) operations like `numpy.fft`, this correspondence between time and frequency samples is hard-coded.) For some of the waveforms you captured above, calculate the power spectrum with $N_{\text{freq}} \gg N$, contrary to the recommendations in §6.2. Use `dft` and choose frequency increments much smaller than $\Delta\nu = \nu_s/N$. Use a logrithmic vertical axis to see if there is nonzero power at frequencies other than ν_0. This is evidence of **spectral leakage**, which affects all power spectra calculated using Fourier techniques.

Can you explain mathematically why you might find power at $\nu \neq \nu_0$ using a Discrete Fourier Transform?

5.5. Frequency Resolution

If you had two spectral lines (sine waves), how closely spaced in frequency could they be and still be resolvable? Investigate this experimentally by combining the output of two function generators in a power splitter. (A splitter run backward is a combiner.) Use two frequencies very close together and plot the power spectrum again using points more closely spaced in frequency than the $\Delta\nu = \nu_s/N$.

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

Can you explain your findings mathematically?

5.6. Nyquist Windows

Above, we calculated spectra for frequencies in the range $\pm\nu_s/2$. What happens if we increase this range?

Explore by taking a Nyquist-sampled time series and calculating the power spectrum for a much larger frequency range, $\pm W\nu_s/2$, where W is at least 4, centered on the original frequency interval. Each value of W gives you a spectrum in a different **Nyquist window**. How do the spectra in different Nyquist windows compare? Note that, for $W > 1$, you are calculating power spectra for frequencies that violate the Nyquist criterion, yet the results

are sensible, in their way. In Lab 4, we will 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—the frequency range of the signal *including negative frequencies*—must not exceed ν_s. For the first Nyquist window this is equivalent to the simpler statement of the Nyquist criterion we first explored.

5.7. Fourier Transforms of Noise

Noise comes into our data from a variety of sources. Here, we restrict the term noise to refer to random fluctations with a known statistical distribution (in our cases, this is almost always a Gaussian with mean zero). We are careful to distinguish this from such things as: systematic contributions to error, bias, ambiguity, and posterior likelihood. Unlike how we use the term colloquially, noise has a specific meaning in science, so be careful with it.

Many of the astronomical signals we observe are, in fact, noise. Blackbodies are noise sources: a blackbody at temperature T emits statistically random electromagnetic waves with specific intensity (power per area per Hz per solid angle) given by the Planck function

$$B_\nu = \frac{2h\nu^3}{c^2}\frac{1}{e^{h\nu/kT}-1}. \tag{1}$$

As radio astronomers, we operate in the regime where $h\nu/kT \ll 1$ (the Rayleigh-Jeans limit), so the blackbody formula simplifies to:

$$B_\nu \approx \frac{2kT}{\lambda^2}. \tag{2}$$

Notice, 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, even if the radiation source is, say, the galactic synchrotron and not even remotely thermal.

Using this framework, all sources of power in our measurement can be related to a temperature, including the receiver noise that is added to our incoming astronomical signal by the thermal motion of electrons in our amplifiers and resistors (otherwise known as **Johnson** noise). This noise (which, by the Central Limit Theorem, is almost always Gaussian) has a variance that is characterized by receiver temperature, T_{rx}, and adds onto the brightness

temperature, T_B, that characterizes the variance of signal coming through the antenna feed of the telescope.

For calibration and testing, we have laboratory sources of noise. Explore the properties of digitally sampled noise:

- Connect the noise generator to a $\sim$ 6-MHz wide bandpass filter (the Minicircuits SBP-21.4 filter) and take a 16000-point time series with the ADC. These samples are voltages. What is the mean voltage over this sample? What is the variance? The standard deviation (which, for a zero-mean signal, is the same as the root-mean-square, or rms)?

- Plot a histogram of the sampled voltages (see `numpy.histogram`—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 observed distribution?

- Take 32 blocks of 16000 samples with the ADC. Compute the power spectrum of each block, as well as the average of all 32 power spectra.

- 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 the relatively smooth distribution that emerges with long integration times. The 'noise' is the scatter you see around that value. How does SNR depend on N? (Hint: SNR is proportional to N^x for some value of x.)

- Calculate the ACF (by manually computing Fourier transfroms and by using `numpy/scipy`'s version) for all 16000 samples in a block. Zoom in on delays of ≤ 2000 samples. Also derive the power spectrum from this ACF and compare with the Fourier-transform-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}). Are they related?

6. Fourier Transforms, Analytic and Discrete (At Home, Week 1)

6.1. The Analytic Fourier Transform

A Fourier transform maps a function between two complementary coordinates which for now are usually time, t, and spectral frequency, ν. The input to a forward Fourier transform is a

signal versus time, $E(t)$; the output is a signal versus frequency, $\tilde{E}(\nu)$, which is computed as

$$\tilde{E}(\nu) = \int_{-T/2}^{T/2} E(t) e^{2\pi i \nu t} dt \ . \tag{3}$$

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

$$E(t) = \frac{1}{B} \int_{-B/2}^{B/2} \tilde{E}(\nu) e^{-2\pi i \nu t} d\nu \ . \tag{4}$$

This works because (complex) sine waves form a **basis** over the space of functions. Any function can be expressed exactly and uniquely by its Fourier coefficients [2]

For those of you looking for applications of that linear algebra class you took, the Fourier transform (FT) is a linear matrix operation akin to a rotation. As such, it has 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.

6.2. The Discrete Fourier Transform (DFT)

The difference between an analytic Fourier transform and a discrete Fourier transform is that signals sampled at discrete intervals are no longer continuous. This means that integrals become somes, and the Nyquist criterion applies.

You have multiple discrete Fourier transform implementations at your disposal. Eventually, we would like you to use `numpy.fft`, which is fast (it can handle millions of points), but does not allow you to choose which frequencies are in your output. It also has the (sensible

[2]You may wonder how the integration limits B and T are defined above. In the proper analytic formulation, they are both infinity. We emphasize their boundedness here because, in practice, neither can be infinity, and this necessarily introduces spectral leakage. 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 the `numpy.fft` and `ifft` conventions.

but confusing) feature that, in the output spectrum puts the zero frequency in the 0th array index, counts upward through the positive frequencies, then switches in the moddle to the most negative frequency and continues counting in the positive direction toward zero. The inverse Fourier transform `numpy.fft.ifft` requires the input spectrum to be in this order to work properly.

However sensible, this arrangement of frequencies is not ideal for plotting. There is a function `fftshift` that will swap the order to a plot-friendly negative-to-positive order. However, once you apply this shift to your array, `ifft` no longer works as you expect. For this reason, *I suggest only using* `fftshift` *in plotting calls to avoid confusion.* See our "DFT's with DFT's" handout for details.

You should also take a look at `fftfreq`, which calculates the frequencies of the `numpy.fft` output.

For this class, we also provide the `ugradio.dft` module. It is slow, but it allows you to manually specify which output frequencies are desired, and it accepts and returns coordinate arrays (time and frequency, respectively) to facilitate clarity and transparency. We recommend using this module in the first week and for applications where you need to oversample the output frequencies. Otherwise, migrate to `numpy.fft`.

To use `ugradio.dft`, here are some recommendations:

- The set of sample times, N. Modern FFT libraries allow for arbitrary N, but performance improves if N has small prime factors. It is also advantageous to use even values of N, so that the zero frequency falls in the center of a bin. Powers of two are common. Our `dft` module is slow, though, and this won't help.
- Define the time range so that $t = 0$ falls at the center of the range. For N even, there is no center time, so make the times run from $-\frac{N}{2}/\nu_s$ to $(\frac{N}{2} - 1)/\nu_s$.
- In specifying the frequencies for which you want the output $\tilde{E}(\nu)$, I suggest that you first calculate the output for N frequencies running from $-\frac{\nu_s}{2}$ to $+\frac{\nu_s}{2}\left(1 - \frac{2}{N}\right)$. This makes the frequency increment equal to $\Delta\nu = \nu_s/N$ over a total range of just under ν_s.

To find out how to use the DFT, you can type `ugradio.dft.<tab>` to see an autocomplete 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 documentation for the DFT function inside the `dft` module.

6.3. 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 '*'),

$$P_\nu = \tilde{E}(\nu)\tilde{E}^*(\nu) \ . \tag{5}$$

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, try casting your array as a 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.

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

Facility with the **convolution theorem** and its cousin, the **correlation theorem**, is a requirement for being a real radio astronomer.

The convolution of two functions E and F (here arbitrarily taken to be functions of time) is

$$[E * F](\tau) = \int_{-T/2}^{+T/2} E(t)F(\tau - t)\ dt, \tag{6}$$

and the correlation of the two functions is

$$[E \star F](\tau) = \int_{-T/2}^{+T/2} E(t)F(\tau + t)\ dt. \tag{7}$$

Conceptually, these two functions describe sliding F over E by changing the delay parameter, τ. The only difference between convolution and correlation 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 3, the convolution theorem states:

$$\widetilde{[E * F]}(\tau) \equiv \int_{-T/2}^{T/2} [E * F](\tau)\ e^{2\pi i \tau \nu} d\tau = \tilde{E}(\nu) \cdot \tilde{F}(\nu) \tag{8}$$

and the correlation theorem:

$$\widetilde{[E \star F]}(\tau) \equiv \int_{-T/2}^{T/2} [E \star F](\tau)\ e^{2\pi i \tau \nu} d\tau = \tilde{E}(\nu) \cdot \tilde{F}^*(\nu) \tag{9}$$

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.

An 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 9 states, in words: **The power spectrum is equal to the Fourier transform of the ACF**

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.

7. Mixers (In the Lab, Week 2)

7.1. The Double-SideBand (DSB) Mixer

In this section, you will build a Double SideBand (DSB) mixer (Figure 1). A mixer is an electronic device that multiplies the two input signals. It is simple: the radio frequency (RF) signal goes into one mixer port, the local oscillator (LO) goes into the second mixer port, and the intermediate frequency (IF) product is output through the third port.

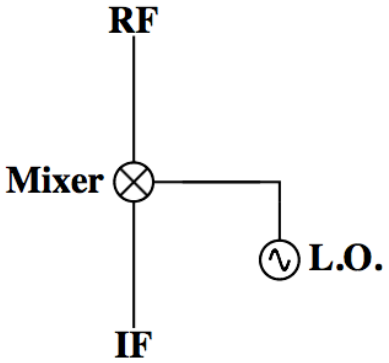

Fig. 1.— A DSB mixer. In the text, we sometimes refer to the RF input as the 'signal'.

Our mixer will be a Mini-Circuits ZAD-1, which has three BNC connectors and works well at these frequencies. The ZAD-1, like nearly all mixers, has its ports labeled "R" (RF), "L" (LO); and "X" or "I" (IF). The ZAD-1 is a balanced mixer, so the R and L ports are identical, and it *will not couple to DC or very low frequencies*. To find out the frequency ranges a ZAD-1 supports on each input, look up its datasheet online.

Use two signal generators as inputs to the mixer and explore the spectra and waveforms in the DSB mixing process. Assign one of the signal generators to be your LO with frequency ν_{LO} and the other your RF "signal" with frequency $\nu_{RF} = \nu_{LO} \pm \delta\nu$. Here, you choose the frequency difference $\delta\nu$ and you set the two signal generators for each of two cases: $\nu_{RF} = \nu_{LO} + \delta\nu$ and $\nu_{RF} = \nu_{LO} - \delta\nu$. Make $\delta\nu$ small compared to ν_{LO}, maybe 5% of ν_{LO}. For the input power level, a good choice is 0 dBm[3] for both synthesizers. The output consists of both the sum and difference frequencies, so choose the ports appropriately.

Digitally sample the mixer output and explore the sum and difference frequencies. As you have learned, there are important issues regarding sampling rate you should get right. Consider 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 $\nu_{RF} = \nu_{LO} \pm \delta\nu$, 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.

7.2. 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?

---

[3]What does this "dBm" mean? It is the power relative to 1 milliwatt, expressed in decibels (dB).

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.

7.3. 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'.

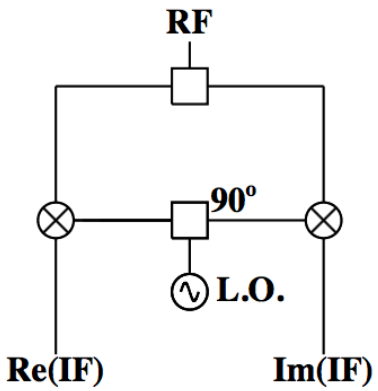

Fig. 2.— An SSB mixer. The important part is the 90° phase delay in the right-hand l.o. This is normally achieved with device called a 'quadrature hybrid'. We will achieve it with a $\lambda/4$ piece of cable.

From the block diagram in Figure 2, construct an SSB mixer that achieves the phase delay with a cable[4]. 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.

7.3.1. 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 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?

7.3.2. 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 §7.3.1. 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?

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

8.1. 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

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

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.

8.2. 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

$$Ae^{iwt} = A\cos(\omega t) + i \cdot A\sin(\omega t). \tag{10}$$

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

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

$$\sin(\omega t) = \frac{1}{2i}(e^{i\omega t} - e^{-i\omega t}) \tag{12}$$

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,

$$Ae^{i(-w)t} = A\cos(\omega t) - i \cdot A\sin(\omega t), \tag{13}$$

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,

$$E(t) = A\sin(\omega_0 - \Delta\omega)t + B\sin(\omega_0 + \Delta\omega)t. \tag{14}$$

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

$$E(t)\cdot e^{-i\omega_0 t} = A\sin(\omega_0 - \Delta\omega)t\cdot e^{-i\omega_0 t} + B\sin(\omega_0 + \Delta\omega)t\cdot e^{-i\omega_0 t} \tag{15}$$

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

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

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

$$\mathrm{LPF}\left[E(t)\cdot e^{-i\omega_0 t}\right] = \frac{A}{2i}e^{-i\Delta\omega t} + \frac{B}{2i}e^{i\Delta\omega t}. \tag{18}$$

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[5].

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

[5]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*.

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.

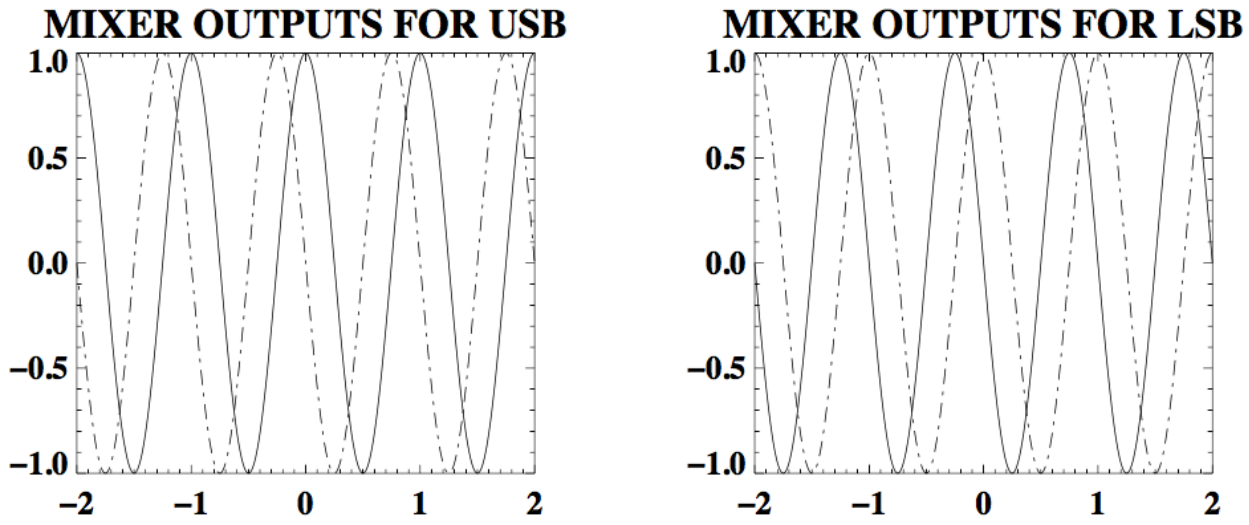

Fig. 3.— Outputs of the first mixers for the two sideband cases. Dashed curve shows the left-hand mixer, solid is the right-hand mixer. Left panel shows $\delta\omega > 0$ (upper sideband—USB); right panel shows $\delta\omega < 0$ (lower sideband—LSB).

8.3. 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 this case, the output of our mixer becomes

$$E(t)\cdot\sin\omega_0 t = A\sin(\omega_0+\Delta\omega)t\cdot\sin(\omega_0 t) \tag{19}$$

$$= \frac{A}{2i}\left[e^{i(\omega_0+\Delta\omega)t} - e^{-i(\omega_0+\Delta\omega)t}\right]\cdot\frac{1}{2i}\left[e^{i\omega_0 t} - e^{-i\omega_0 t}\right] \tag{20}$$

$$= \frac{A}{2}\left[e^{i\Delta\omega t} + e^{-i\Delta\omega_0 t} - e^{i(2\omega_0+\Delta\omega)t} - e^{-i(2\omega_0+\Delta\omega)t}\right] \tag{21}$$

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

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

$$E(t) \cdot \sin\omega_0 t = A\left[\cos\Delta\omega t - \cos(2\omega_0 + \Delta)t\right] + B\left[\cos(-\Delta\omega)t - \cos(2\omega_0 - \Delta)t\right]. \qquad (23)$$

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

$$\mathrm{LPF}\left[E(t) \cdot \sin\omega_0 t\right] = (A + B)\cos\Delta\omega t. \qquad (24)$$

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.

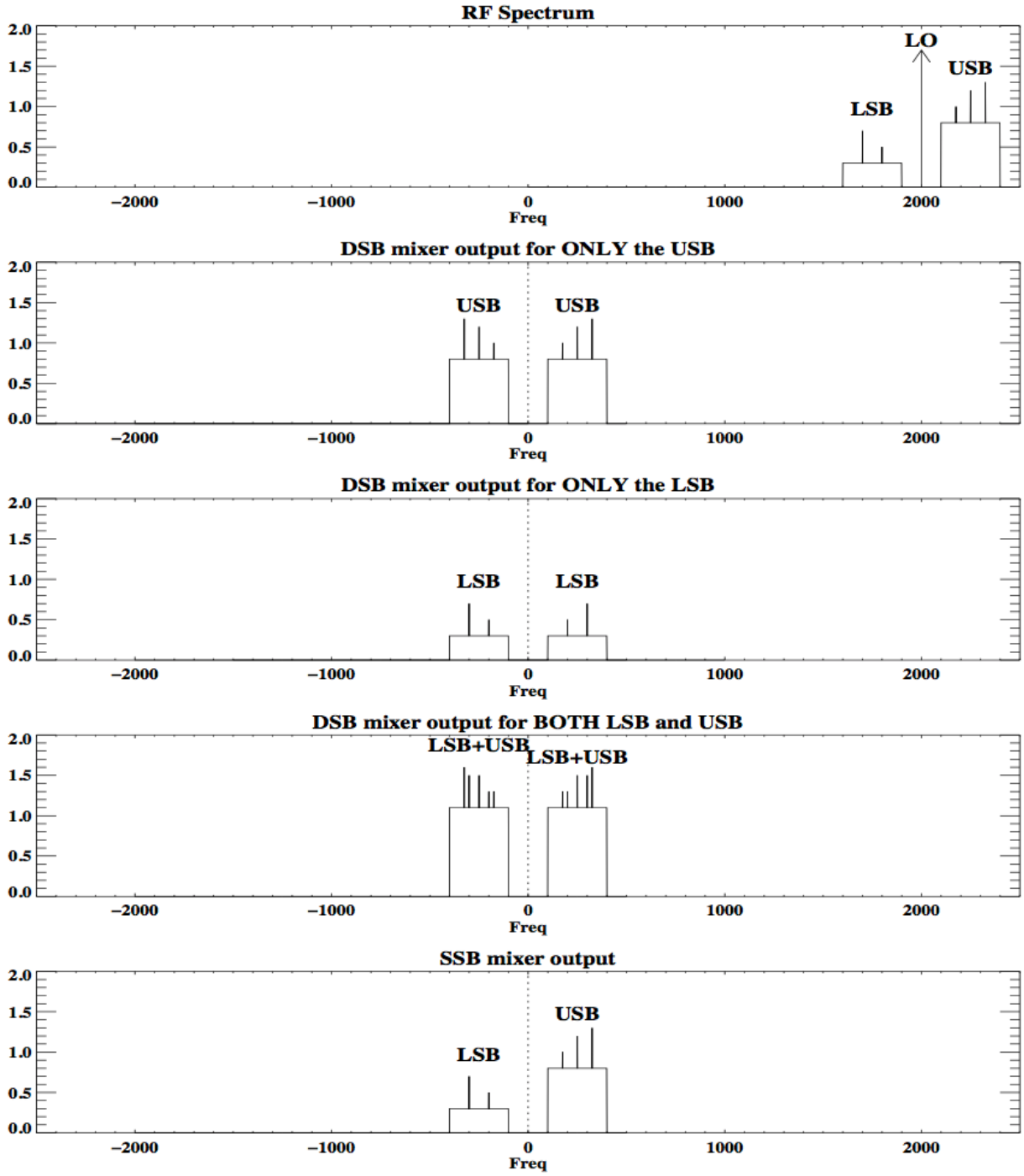

Fig. 4.— Upper and lower sidebands in DSB and SSB mixers for a set of δ-function test signals on top of broad level noise spectra. Top: the RF spectrum. The next two show the USB and LSB individually when they undergo the DSB mixing process; panel 4 shows how they both add together. The bottom panel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

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.

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

9.1. 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.