Project 2: Processing Pipeline

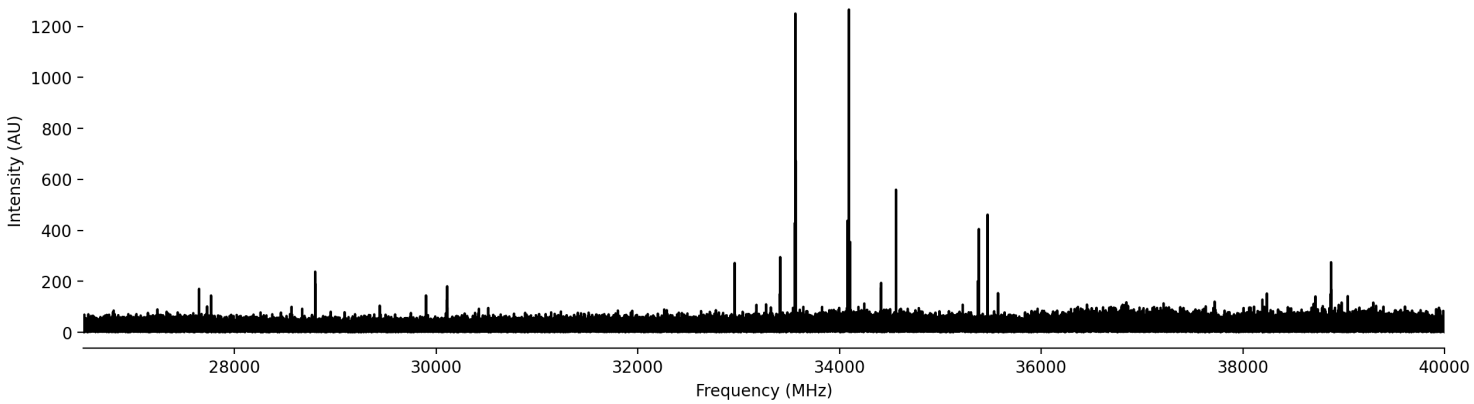
Start Assignment

Due Monday by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip, ipynb, py, and html

In this assignment, you will create a data processing/visualization pipeline for an experimental data set. You will be working with data from an experimental instrument in the Crabtree lab: a chirped-pulse Fourier transform microwave spectrometer that operates in the 26.5-40 GHz frequency region. Like NMR, the raw data from the instrument is a free induction decay that needs to be processed with a Fourier transform, revealing the peaks in the frequency spectrum. You will find the data in a csv file called data.csv in the **Project 2 Folder** in the Files section of Canvas. Your objective will be to write a processing pipeline that performs the Fourier transform and analyzes the peaks in the spectrum. The steps you need to perform are shown below, along with graphs showing what your output should look like if your code is working correctly. **Your plots do not need to be formatted identically to these; you may customize them as you wish.** These are here to show you in general what kinds of plots to make and approximately what the data should look like.

1.) Preprocess the FID and perform the FT

The raw data contains a 15 microsecond-long FID with the first ~3 microseconds zeroed out. To process it, the FID first needs to be multiplied by a Blackman-Harris window function (see **scipy.signal.get_window (https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.get_window.html)**), then the Fourier transform needs to be computed (see **scipy.fft.rfft (https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.rfft.html)** and **scipy.fft.rfftfreq (https://docs.scipy.org/doc/scipy/reference/generated/scipy.fft.rfftfreq.html)**). Because of the way the instrument is constructed, the frequency you calculate for the FT x axis needs to be subtracted from 40,960 MHz (i.e., 40,960 MHz - f). Although the data will contain a wider frequency range, only the frequencies between 26,500 and 40,000 MHz are important. Create a plot to visualize the FT in this region. It should look something like this:



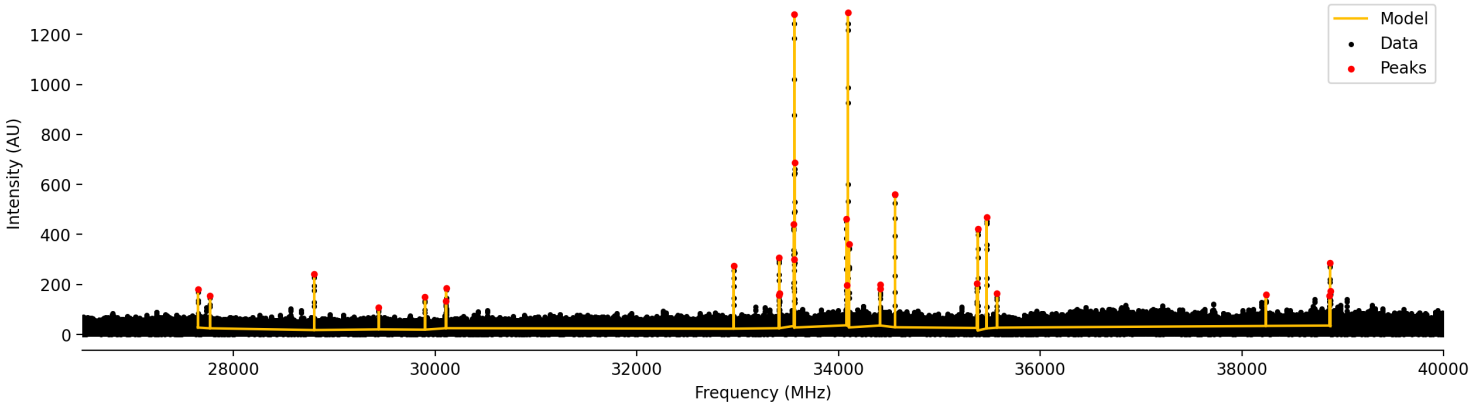
2.) Fit the peaks to a Gaussian lineshape function

The spectrum contains many peaks. Each one has approximately a Gaussian lineshape, and is sitting on top of some baseline noise:

$$f(x) = y_0 + A\exp\left(-\frac{(x-x_0)^2}{2w^2}\right) f(x) = y_0 + A\exp\left(-\frac{(x-x_0)^2}{2w^2}\right)$$

where y_0 is the noise level, A is the peak height, x_0 is the center frequency of the peak, and w is the standard deviation (related to the peak width). Use **scipy.optimize.curve_fit (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html)**

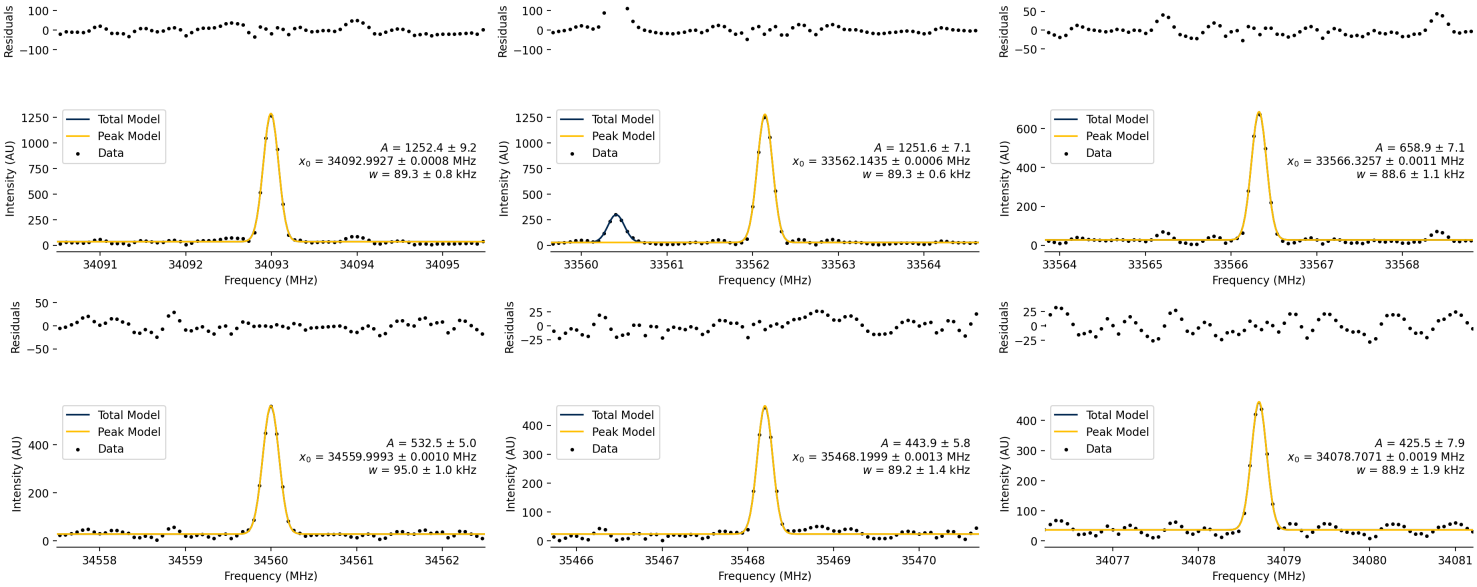
to fit each peak in the spectrum with an appreciable signal-to-noise ratio (I chose peak height divided by noise level > 7) to this Gaussian function. To do this, you will need initial guesses for the model parameters. You will need to figure out how to come up with these guesses! Consider writing or using an existing peak finding algorithm (**scipy.signal.find_peaks (https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html)** might be a good place to start). Create a plot that shows the model function on top of the experimental data, with each peak "tagged." It should look approximately like this (you may have a couple more or a couple fewer peaks, depending on what threshold you use):



As a hint for peak detection and fitting: the FID can be zero padded to increase the number of data points in the FT. Zero-padding is simply adding a bunch of 0s to the end of the FID before the FT is computed. The FFT algorithm is especially fast when the data length is a power of 2, so you might try calculating FTs from FIDs that have been zero-padded to a length of 2^{20} or 2^{21} points to help you find and/or fit peaks.

3.) Create a report of the results

Create a csv file containing the model parameters for each peak and their 1 sigma uncertainties. I've uploaded a peaks.csv file as an example of what the output should look like. In addition, for the strongest 6 peaks in the spectrum, make a plot of your model and the data zoomed in around a +/- 5 MHz region around the peak, along with the residuals. Annotate the plot with the A, x0, and w parameters for the peak. An example is below. Note that sometimes multiple peaks may be close together (See the top middle plot)! I'm showing both the entire model fit and the fit for only the strong peak. The residuals are calculated with only the strong peak.



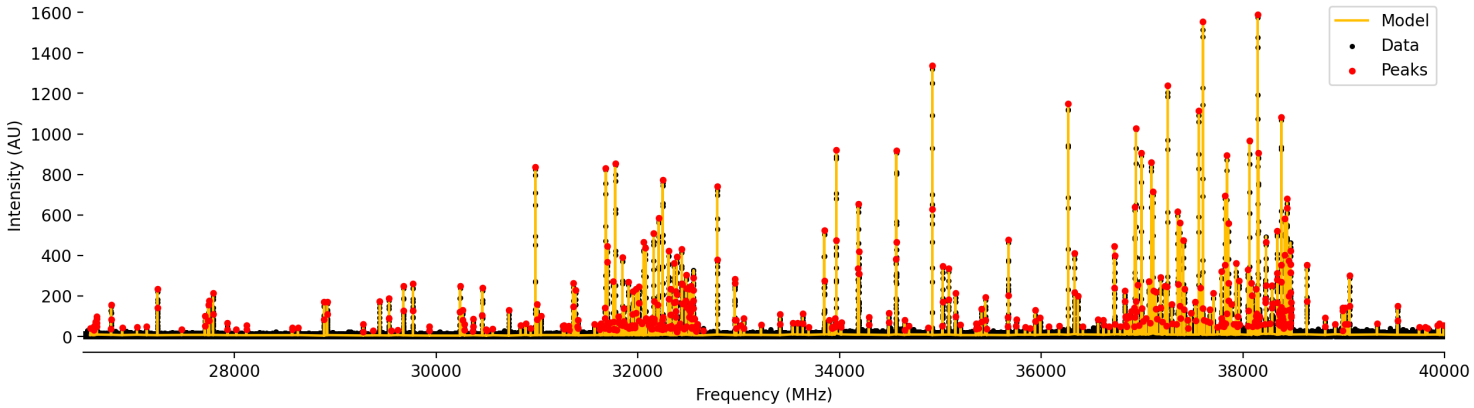
4.) What to submit, and testing your pipeline

You should turn in an ipynb file (with all outputs cleared) that performs the steps above and produces the requested output (3 graphs and a csv file) in a single run, along with an HTML version that shows all of the cell outputs.

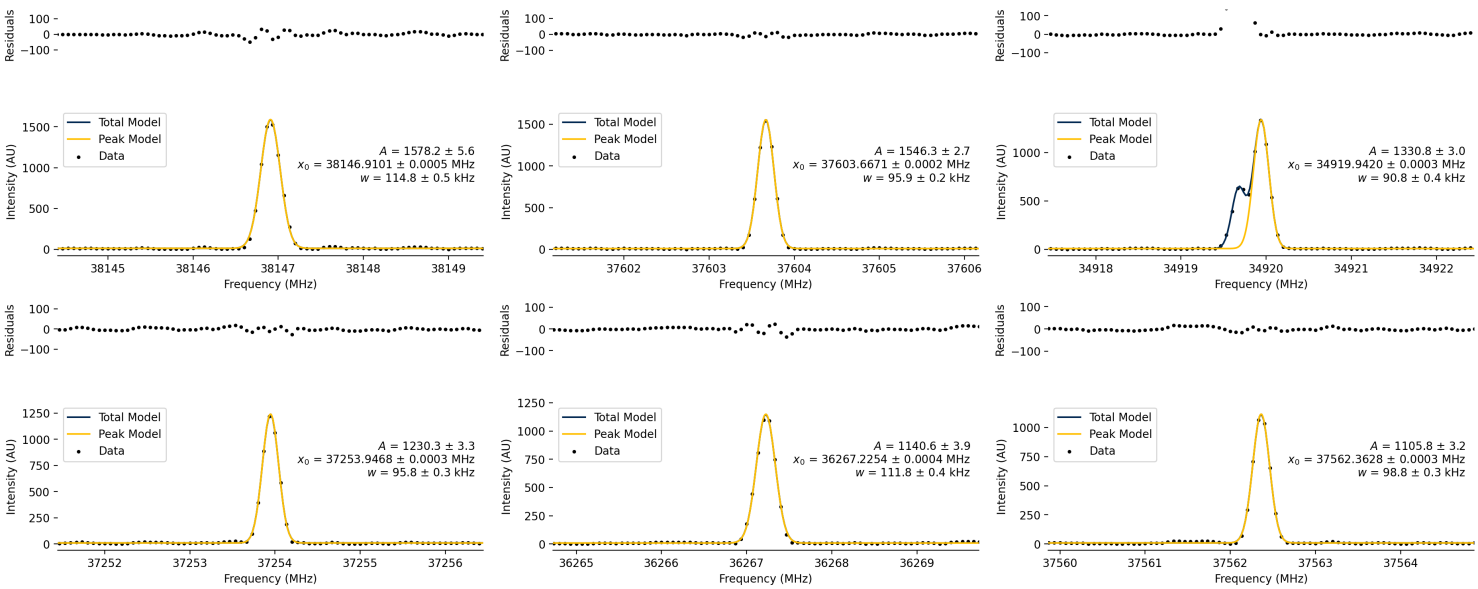
To put your pipeline to the test, we will run your code on a different csv file containing data from the same instrument. For full credit, your code must process this file successfully too. The FID will have the same general structure, but all of the frequency peaks will be different. Make sure you're not hard-coding anything specific to the example spectrum!

5.) (Optional) More of a challenge!

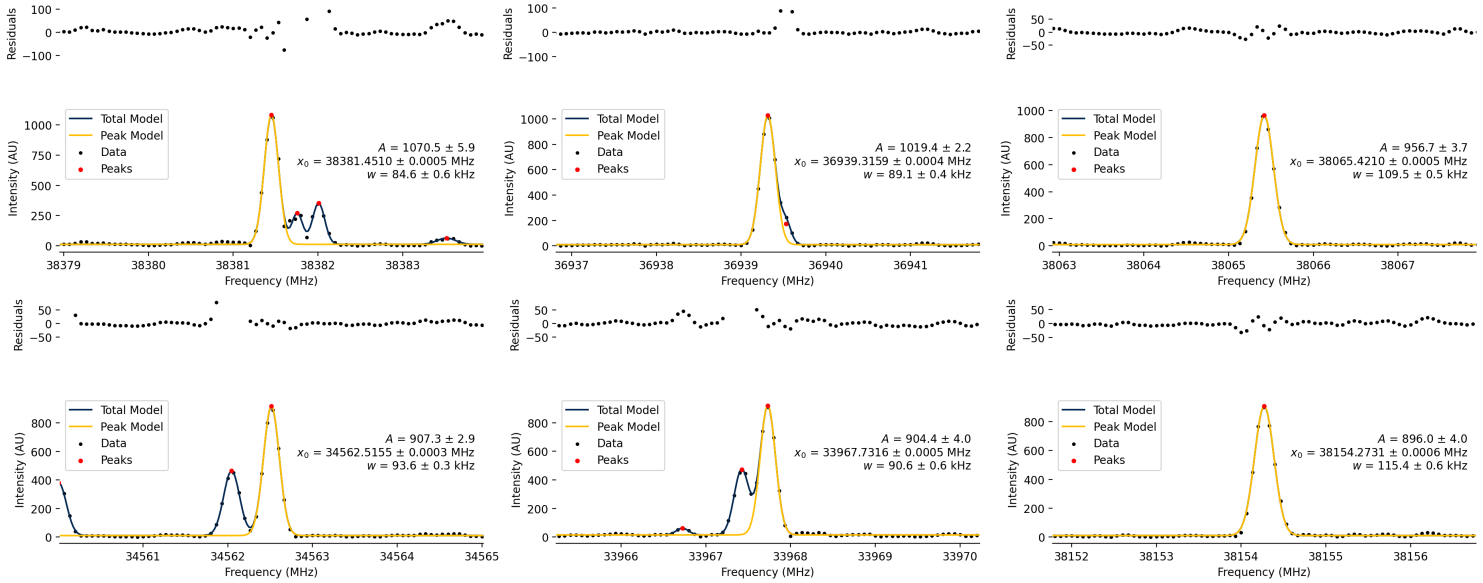
There is also a challenge.csv file. This one contains the spectrum of pyridine, which is much more complicated than the data your code will be tested on. It has hundreds of peaks, and many of them are blended (I'll show some examples below). If your pipeline can handle pyridine, then it should have no trouble with the testing data we use when we grade. Here is an overview of the spectrum, showing my pipeline's model and peaks.



And if we take a look at some of the peaks, we can see that there are some complications. Here are the 6 strongest peaks:



Notice how on the upper right plot there are 2 peaks blended together. This might present a challenge for your pipeline. But it gets even better! Here are the next 6 strongest lines, and I've added markers to show where the individual peaks are:



Notice especially the top center: there are 2 peaks blended so strongly that there is not even a local maximum for the smaller peak! There are several peaks like this in the data set. If your pipeline can work automatically for that case, then you've done a great job. However, it is not required that you try to solve this challenge for the assignment. The test data do have nearby lines, but they will not be as strongly blended as these.

FTMW Pipeline Rubiric

Criteria	Ratings					Pts
Preprocess FID and FT This code was given in class.	**5 pts** **Excellent** Blackman-Harris window function is correctly applied to the FID, and the FT is plotted with the correct frequency range of 26.5-40 GHz.	**3 pts** **Very Good** FT is performed, but window function is not applied correctly. Minor errors in frequency of spectrum. Minor plot issues (titles, axis labels, etc)	**2 pts** **Good** Performed FT, but plot is lacking in any labels. Window function is not applied.	**1 pts** **Fair** Serious flaws in FT.	**0 pts** **No Marks** No FT was performed	5 pts
Fit the peaks to a Gaussian	**10 pts** **Excellent** Peaks are found using a cutoff based on SNR. Each peak in the list is fit using a gaussian line shape, and the parameters are ploted with the spectrum as a "model function" on top of the experimental data.	**8 pts** **Very Good** Peaks are found and fit, but some minor flaws in the plotting, or in the development of the model function.	**6 pts** **Good** All peaks are identified, and the fit is attempted but unsuccessful.	**4 pts** **Fair** Peak finder used, but too many, or too few peaks are identified	**0 pts** **No Marks** No peaks were found or fit	10 pts
Report of the results	**10 pts** **Full Marks** CSV file is produced with model parameters for each peak and their 1 sigma uncertainties, with the same, or nearly the same values expected. Plots produced of 6 strongest peak are a 10 MHz window around the peak, and are graphed together with the residuals.	**8 pts** **Very Good** CSV file is produced, with close to the correct parameters for the gaussian fit, with minor errors. Plots are produced for the 6 largest peaks, but with minor flaws in the presentation.	**6 pts** **Good** No CSV file or no plots are produced, but the other is high quality.	**4 pts** **Fair** Major issues with the csv file and the 6 plots.	**0 pts** **No approval** No report or plots produced	10 pts

Criteria	Ratings						Pts
Execution How well does the python code run?	**10 pts** **Excellent** Code is well-written, making use of efficient routines for fast computation. Code runs and produces all required output without errors. Any warnings are explained/handled. Code works with no modification to workup different dataset.	**8 pts** **Very Good** Code is well-written and mostly fast/efficient. No significant errors, and any warnings are explained/handled. Some work is required to get the data to work with a new dataset.	**6 pts** **Good** The code gets the job done, though could be improved by making use of more efficient coding patterns. Code may contain an error that is easy to fix (e.g., cells ordered incorrectly). Code requires major changes to analyze a different dataset.	**4 pts** **Fair** Code is slow/inefficient, or the code contains mistakes that cause the output to be incorrect or unintended. Code does not work with a different dataset	**2 pts** **Poor** Code is slow/inefficient and/or cannot be successfully run from beginning to end without significant errors.	**0 pts** **No Marks** Code does not run at all and contains major errors in construction.	10 pts
Readability/Documentation Note that code comments can be in the form of inline python comments or markdown annotations in a jupyter notebook (or a combination of the two)	**15 pts** **Excellent** Code is well-organized, easy to understand, and well-commented. Variables/functions/classes have names that make clear what their purpose is, and any especially tricky areas of the code include detailed comments that explain the intended function.		**10 pts** **Good** Most parts of the code are organized and easy to read for someone familiar with the project. Variables/classes/functions are named appropriately. Comments are mostly informative for explaining what is happening.		**5 pts** **Poor** Code is difficult to follow even for an experienced python programmer. Variables/functions/classes may have inappropriate names. Comments are sparse.	**0 pts** **No Marks** Code seems obfuscated and contains almost no comments.	15 pts
						Total Points: 50	