Fourier Methods

Johnny Pribyl

March 9, 2018

Abstract

This lab is designed to familiarize the student with thinking in the frequency domain. It requires far more data collection than the previous labs. There are four main objectives. First, we worked with a few simple input functions in order to become familiar with the spectrum analyzer. Second, we investigated the response of an LRC circuit to several different signal types. Third, we did the same thing to an acoustical cavity. And lastly, we examined a system of coupled torsional reed oscillators. Unfortunately, we ran out of time and were not able to collect all of the necessary data from the reed oscillators - so, this report will only include an informal discussion of theory and observed results.

1 Introduction

Somehow, every undergraduate physics student manages to sneak through their classes without ever fully understanding Fourier methods. Brian took it upon himself to fill this educational hole with experimental knowledge. That means this lab operates almost entirely in the frequency domain. Or, in other words, we Fourier Transform *everything*.

Although it's pretty tough to get funding these days, MSU still has some toys leftover from the 90s. Among those is the SR770 Spectrum Analyzer. It boasts a peaceful green-hued digital output and the capacity to live stream the frequency domain of a time-dependent signal. It only lacks the phrase ***DON'T PANIC.*** Ideally, this would be inscribed in large friendly letters on its cover.

We used the SR770 and TBS1052B-EDU oscilloscope extensively in this lab. Both of them are capable of saving hundreds of data points at the press of a button. So, we had quite a bit more real data to analyze for this lab than we have had for any of our previous forays.

As before, you can find all of my code and data at:

`jpribyl/cautious-palm-tree`

1.1 Database Design

The majority of this lab's data is relatively straightforward. Typically there are two columns - one for the dependent variable (say, voltage) and one for the independent variable (typically time or frequency). We saved this data into .csv files on a USB. However, all of the measurements carry a large amount of metadata regarding input frequency, amplitude, channel, and the source of the data. Unfortunately, all of the metadata is not recorded into the files. We wrote it all down in our lab notebooks, but that makes it difficult to cross reference.

It would definitely be possible to store all the time / voltage data in excel and assign values to the metadata with python as we read it in. However, that approach does not scale very well. It's particularly ineffective for collaborative efforts. Ideally, there would be a way to tie all of the metadata to the files themselves. This ensures data integrity without requiring all collaborators to use an identical code base. It turns out, I'm not the first person in the world who has had this problem. The correct tool for the job is a "database."

Databases are created and maintained with a language called SQL, or Structured Query Language. Every measurement is given a unique identifier that allows anyone to access or "query" all of its metadata.

Our data did not come pre-packaged with unique identifiers, so I had to add them manually. Bash has several incredibly powerful text-editing commands that are perfect for this kind of task. I'm not as familiar with Awk, so I opted to use Sed. If you have a bash shell on your computer you should be able to access the man pages with

```
man sed
```

Otherwise, you can read them online at gnu.org/software/sed/manual/sed.txt. The first step was collecting all of our files into a single directory and naming them numerically so that I can sort through them with a for loop:

```bash
#!/bin/bash
cur_id=1;

for i in $(ls * | sort -V);
do
    echo $i
    echo $cur_id

    #double quotes allow variables
    sed -i "s/\(.*\)/$cur_id,\1/" $i

    let cur_id+s=1
done
```

I find bash incredibly hard to read, so let's break this down a bit. The very first line is known as a "shebang" and tells the computer to use bash for this script. Next I define a variable "cur_id" which holds the unique identifier that will get prepended to every line of the data files.

After that I run through every file in the current directory and sort them "naturally." For each file, I read out (or echo) the file name and the file's identifier. Then, I tack the current id and a comma in front of every line in the file and increment the cur_id by 1.

Let's say that the first file in a given directory looks like this:

```
0.0000000e+000, -1.3732910e-001,
6.2500000e+001,  9.4042969e+000,
...
1.2500000e+002, -4.9438477e-002,
1.8750000e+002, -7.1411133e-002,
2.5000000e+002, -3.2958984e-002,
3.1250000e+002, -3.2958984e-002,
```

My script will turn it into something that looks like this:

```
1,  0.0000000e+000, -1.3732910e-001,
1,  6.2500000e+001,  9.4042969e+000,
...
1,  1.2500000e+002, -4.9438477e-002,
1,  1.8750000e+002, -7.1411133e-002,
1,  2.5000000e+002, -3.2958984e-002,
1,  3.1250000e+002, -3.2958984e-002,
```

Where 1 is the unique identifier for this measurement. The second file in the directory would look very similar except that it would be prepended with a 2 instead of a 1.

After pre-processing the data, I read it into my database. If you want to play with the database at home, you can restore it using MySQL Workbench from the dump files in

```
lab3/data/database/databasedump
```

If you don't feel like downloading my database, I really wouldn't blame you. It's not exactly anything groundbreaking. Besides, I took the 30 seconds required to generate an EER diagram mapping the relationships between tables. I'll let you peruse it at your leisure.

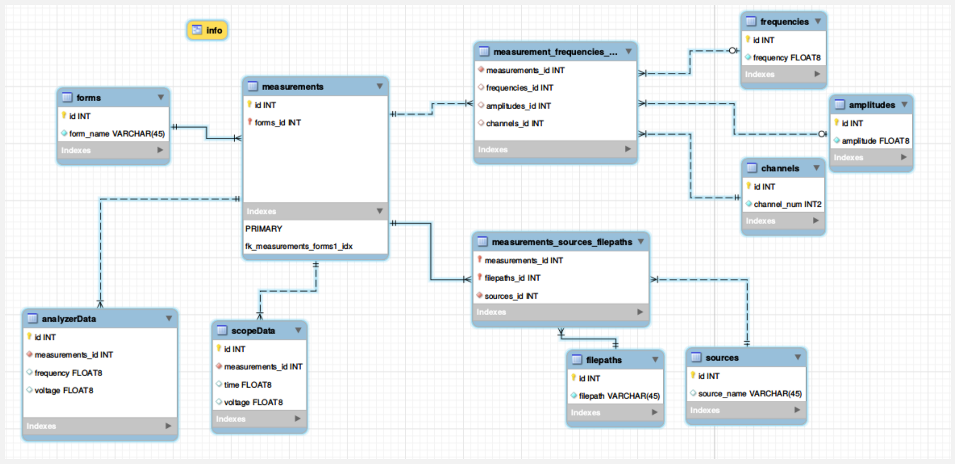
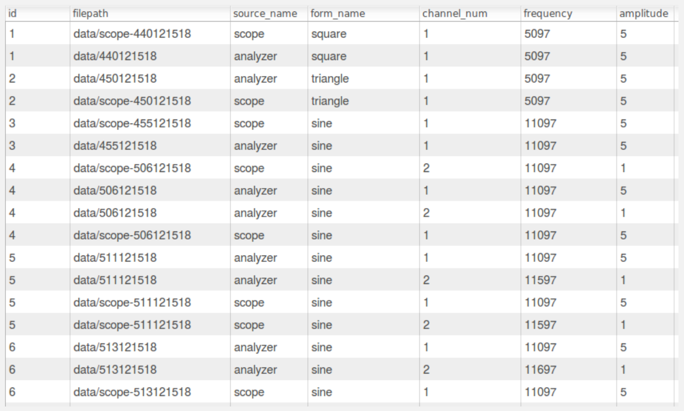


Figure 1: EER diagram mapping the database

The little yellow box entitled "Info" corresponds to a view, or saved query that allows for easier access to the desired data without sacrificing the structure of an ACID database. If you were to run a simple query on the view:

```
select * from info
```

You would get something back that looks like:



id	filepath	source_name	form_name	channel_num	frequency	amplitude
1	data/scope-440121518	scope	square	1	5097	5
1	data/440121518	analyzer	square	1	5097	5
2	data/450121518	analyzer	triangle	1	5097	5
2	data/scope-450121518	scope	triangle	1	5097	5
3	data/scope-455121518	scope	sine	1	11097	5
3	data/455121518	analyzer	sine	1	11097	5
4	data/scope-506121518	scope	sine	2	11097	1
4	data/506121518	analyzer	sine	1	11097	5
4	data/506121518	analyzer	sine	2	11097	1
4	data/scope-506121518	scope	sine	1	11097	5
5	data/511121518	analyzer	sine	1	11097	5
5	data/511121518	analyzer	sine	2	11597	1
5	data/scope-511121518	scope	sine	1	11097	5
5	data/scope-511121518	scope	sine	2	11597	1
6	data/513121518	analyzer	sine	1	11097	5
6	data/513121518	analyzer	sine	2	11697	1
6	data/scope-513121518	scope	sine	1	11097	5

Figure 2: A simple query on the database

Hopefully all of that made sense. If not, don't worry about it – it's not really too relevant to the actual data analysis.

2 Objectives

All of these labs have objectives instead of procedures. They are intended to guide our experiments without spoon feeding us the results. Sometimes I really miss eating bananas out of a jar. Fortunately for me, most grocery stores still stock Gerber snacks.

If you're not familiar with the methods and procedures of this lab, then I would suggest reviewing the manual. It is quite extensive and details all the theory far more eloquently than I ever could. It lives in:

```
lab3/lab_descrip/Fourier_Methods_manual.pdf
```

2.1 Introduction to Spectrum Analyzers

2.1.1 Sine, Square, and Triangle Waves

The first thing we did in lab after putting our heads on straight was run a few single frequency waves through the oscilloscope and spectrum analyzer. The goal of this section was to get comfortable with the conversion between frequency and time domains. All of the oscilloscope data lives in the time domain. This means that time is the independent variable and our plots consider $Voltage(time)$. However, it is often useful and always didactic to convert this data into $V(\omega)$ by Fourier Transforming it:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt \tag{1}$$

Technically, we have a discreet data set that isn't quite infinite, so it might be more accurate to say that we are using a Fourier Series – but the theory is extremely similar and a full discourse on Fourier Analysis is outside of the scope of this report.

If we use the function generator to produce a Sine wave of 11,097 Hz, we get a familiar looking graph of $V(t)$ on the scope:

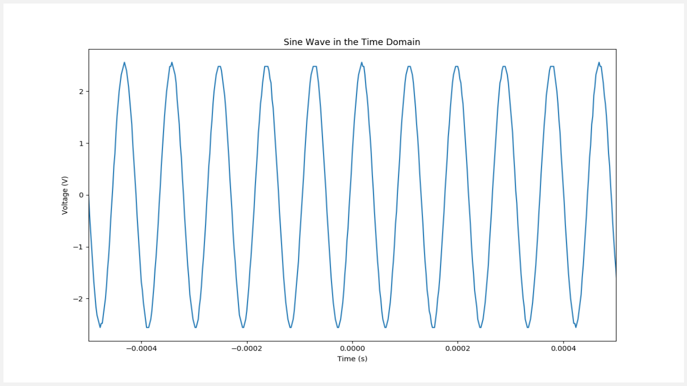


Figure 3: Sine wave of 11.097 kHz collected from the oscilloscope

Now, everyone knows that the Fourier Transform of a sine wave is a delta function so let's transform the data that we collected from the scope and see how it looks:

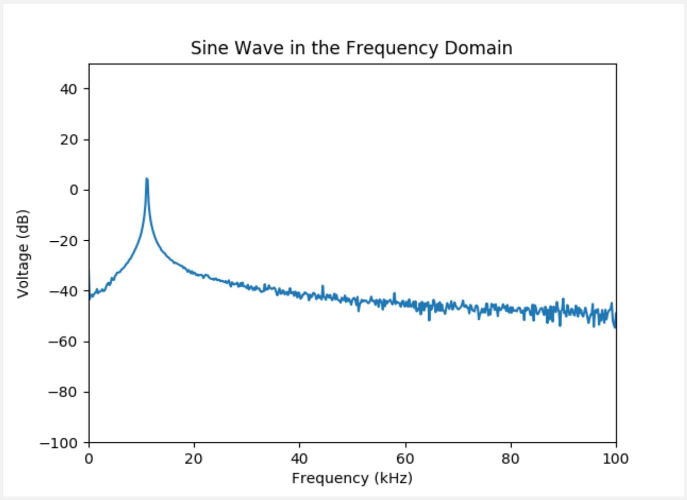


Figure 4: FFT of data presented in Figure 3

So far, it seems like all is well in the world, but let's overlay the results with the data collected from the SR770 and see whether things actually match up.

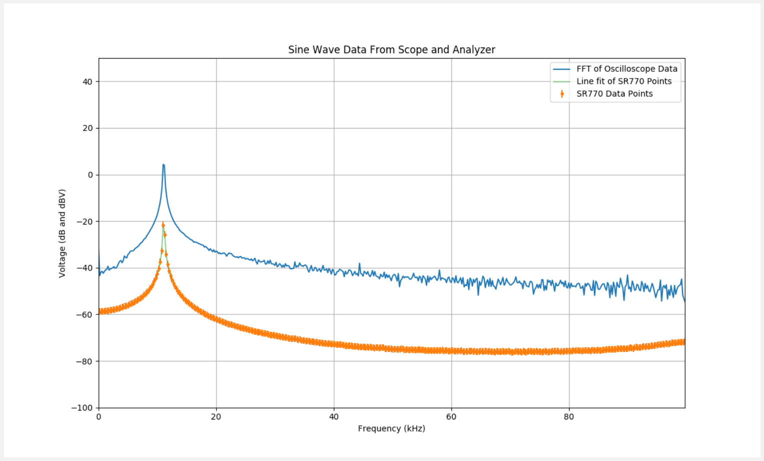


Figure 5: Analyzer data overlaid on top of Figure 4

Things are looking pretty good – however, those amplitudes don't quite seem to match. At a glance that might seem problematic. But, if you look closer at the label on my y-axis, you'll notice that the units are not the same! The data from the SR770 is in dBV while the data from the oscilloscope is in dB. In fact, if you convert everything to a linear scale, multiply by the amplitude of the input current, and convert back to a log scale – the amplitudes just about match up:

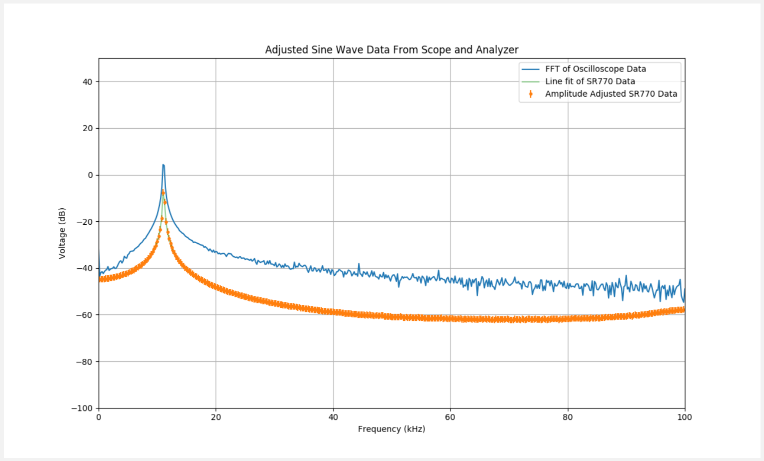


Figure 6: Amplitude matching by converting dBV to dB

The last thing to do before moving on is model the results. The plot is starting to get a little bit loud, but modeling a sine wave is quite easy. All we have to do is plug in the frequency, transform a vector of points, and overlay the final plot in the series

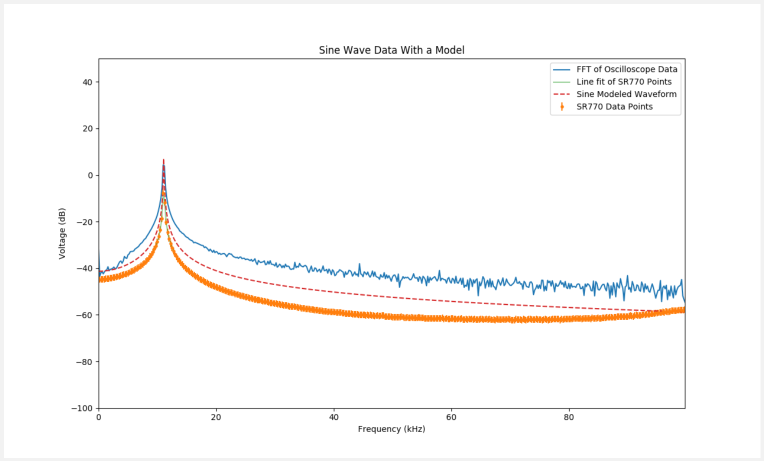


Figure 7: Integrating a theoretical model to Figure 6

The coding aspect of this lab was extensive enough that it seemed to warrant utilizing classes. So, I defined a class for measurements. The attributes for this class could be a useful reference, so here they are:

```
    :measurementId: Int - unique identifier

    :an: Pandas DF - holds all analyzer data

    :xan: Pandas DF - uncertainties and values in x
    :yan: Pandas DF - uncertainties and values in y

    :xanvalues: Pandas DF - values for x component
    :yanvalues: Pandas DF - values for y component

    :xanerror: Pandas DF - uncertainties for x component
    :yanerror: Pandas DF - uncertainties for y component

    :sc: Pandas DF - holds all oscilloscope data

    :xsc: Pandas DF - holds values for x component
    :ysc: Pandas DF - holds values for y component
```

I can't say that I'm especially proud of the class method for taking a Fourier Transform, but it does get the job done.

This method is available to any instances of the measurement class:

```python
    def fourierTransformVoltage(self):
        """
        :returns: nothing - however, will fourier transform,
        normalize, convert voltage to dB, and set all the
        necessary variables for plotting
        """

        try:
            #pick out values and time step
            self.voltage = self.voltage.apply(lambda x: x.n)
            self.time = self.time.apply(lambda x: x.n)

            #perform the transformation on voltage
            fftvolt = np.fft.fft(self.voltage)

            #clean up and normalize
            fftvolt = np.fft.fftshift(fftvolt)
            fftvolt= 2*fftvolt/float(len(self.voltage))

            #convert voltage to dB
            self.fftdbv = 20.*np.log10(np.abs(fftvolt))

            #determine the time step and window length
            self.time_step = self.time[2]- self.time[1]
            self.win_length = len(self.time)

            #recover frequency bins
            self.fftfreq = np.fft.fftfreq(
                self.win_length, self.time_step)

            self.fftfreq = np.fft.fftshift(self.fftfreq)

            #convert to kHz
            self.fftfreq = self.fftfreq/1000.

            self.xsc = self.fftfreq
            self.ysc = self.fftdbv

        #all code to contine if fft fails
        except Exception as e:
            print('could not perform FT: ', e)
```

The data that we collected is available in data/magnetic.xlsx and my lab notebook, so I will not recopy it here. But, here is what it looks like:

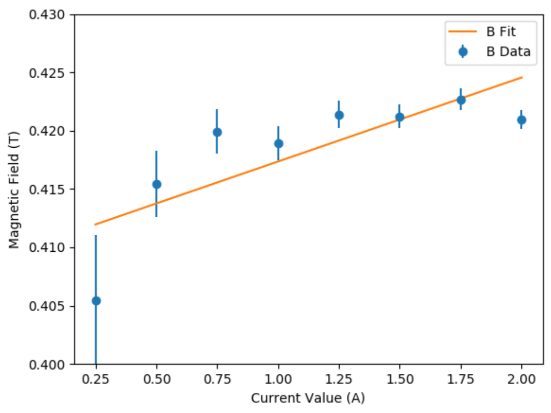


Figure 8: Plotting the Measured values for Magnetic Field

I chose to read my data into python as a Pandas DataFrame. In my experience, pandas is just about the best library to work with heterogeneous data. It's even able to read an excel file:

```
xl = pd.ExcelFile('data/magnetic.xlsx')
df = xl.parse('Sheet1')
```

After that, I dropped the non-existent entries off the bottom of the DataFrame because they're problematic for model fitting. Then, I used the uncerainties package and a lambda function to propagate error through all of our mass and current measurements. I'm not going to copy all of the code here, but the general syntax follows this form:

```
<Measurement>= \
    df[<Measurement>].dropna().apply(
        lambda x: ufloat(x, <error>)
    )
```

Next, for the current menasurements, I looked up the specs for Keithley's model 2000 6 1/2 digit multimeter. I stared at them for a while. Then I pestered Brian for a while. Then I stared at the specs some more. Eventually, I pestered Brian enough that he showed how to read the table. In our case, the current has an uncertainty of:

$$(1000 \times I + 3 \times 15) \times 10^{-6}$$

In python, we are able to make use of the pandas data structure and uncertainties library to propagate this:

```
current_error = \
    1000 * df['current'].dropna() * 10 ** -6 + 3 * 15 * 10**-6

b_cal_current = \
    pd.Series(uarray(df['current'].dropna(), current_error))
```

Notice that I have to explicitly turn the result back into a pandas object. The uarray() method returns a NumPy Array. It would be totally fine to leave the result as a NumPy object, but syntactically NumPy is slightly different and Pandas, so it's beneficial to have all objects be the same type.

I was able to fit the curve using the same method as in the data analysis lab. Specifically, I assumed linearity and fit it with:

```
def lin_fit(x, a, b):
    return a*x + b

popt, pcov = curve_fit(
                lin_fit,
                current_values,
                b_cal_values
            )

b_fit = lin_fit(
    current_values,
    *popt
)
```

We learned last lab that residuals are a pretty decent sanity check on the accuracy of data and models. So let's go ahead and plot the residuals from this fit:

```
r_i = b_cal_values - b_fit

plt.errorbar(
    current_values,
    r_i,
    yerr=b_cal_error,
    fmt='o')
```

And showing this plot, we find that the residuals are actually quite reasonable. They are clustered around zero and their error bars are easily visible. Notice that the size of the error bars around zero is quite large. This makes sense because the current error ought to be similar in magnitude, but its fractional error will increase:

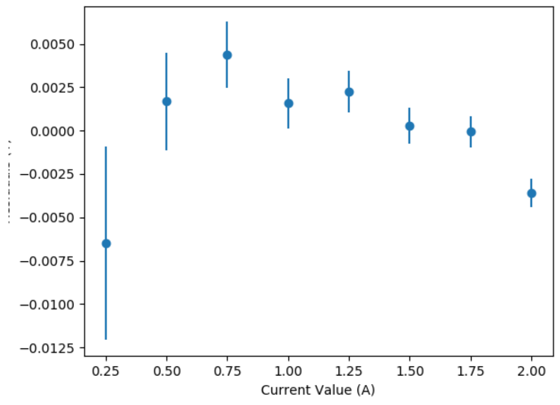


Figure 9: Plotting the residuals as a sanity check

2.2 Magnetic Susceptibility of Samples

Now, hopefully I've convinced you that the environment between our magnets houses a magnetic field that is approximately .4 Teslas. If I haven't convinced you yet, try this:

```
def understand(paper, confused=True):
    if confused:
        read(section_1)
        read(section_2)
        understand(paper)

    return
```

Moving right along, We massed all our samples using the Guoy balance. Then, we did some math to determine how much of our sample was empty space and how much was really the substance in question. For example, if you have a vial full of pebbles, then a good bit of the vial is actually just air. The actual area is given by:

$$A = (w \times l)(\%real)$$

Where % real is:

$$\frac{m_{measured}}{m_{theoretical}}$$

And, for a substance of known density ρ, theoretical mass is:

$$volume \times \rho$$

Putting all of that together with the equation that we derived in lecture, we find the magnetic susceptibilty with:

$$\chi_m = \frac{2 \times \mu_0 \times g(m_1 - m_0)}{AB^2}$$

As before, m_1 refers to the mass of the magnet apparatus with the sample sitting in between the magnet and m_0 is the mass of the magnet without a sample in between it.

One peculiarity that I had not considered prior to doing my data analysis was the possibility of getting a value larger than 1 for % real. This does not physically make sense (because our samples were not pressurized). For most of the samples, this was not an issue.

However, for the two wire samples, things got a little interesting. The wires were too small for us to measure very accurately which led us to accumulate immensely large errors. After thinking on the issue for a while, eventually I decided to accept the measurements on these wires that are provided in the lab to be exact. I also decided to refuse negatve percentages and cap the maximum % real at 1.

In python, we can impose reaonability on percentages with:

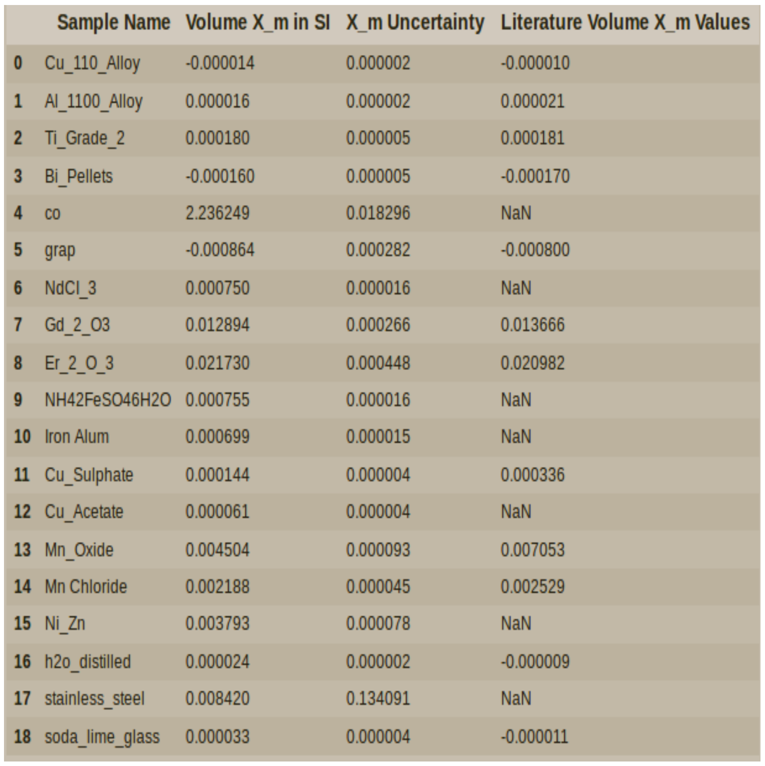
```
percent_real = (real_mass / theory_mass)

percent_real.apply(
    lambda x: ufloat(min(abs(x.n), 1), x.s)
)
```

Then, we can drop the error from cobalt:

```
area[4] = area[4].n
percent_real[4] = percent_real[4].n
```

And now we're off to the races! Doing all the math things and putting them in a table thing that compares measurements and uncertainties to accepted χ_m values, we see that most of our results are actually quite good:



	Sample Name	Volume X_m in SI	X_m Uncertainty	Literature Volume X_m Values
0	Cu_110_Alloy	-0.000014	0.000002	-0.000010
1	Al_1100_Alloy	0.000016	0.000002	0.000021
2	Ti_Grade_2	0.000180	0.000005	0.000181
3	Bi_Pellets	-0.000160	0.000005	-0.000170
4	co	2.236249	0.018296	NaN
5	grap	-0.000864	0.000282	-0.000800
6	NdCl_3	0.000750	0.000016	NaN
7	Gd_2_O3	0.012894	0.000266	0.013666
8	Er_2_O_3	0.021730	0.000448	0.020982
9	NH42FeSO46H2O	0.000755	0.000016	NaN
10	Iron Alum	0.000699	0.000015	NaN
11	Cu_Sulphate	0.000144	0.000004	0.000336
12	Cu_Acetate	0.000061	0.000004	NaN
13	Mn_Oxide	0.004504	0.000093	0.007053
14	Mn Chloride	0.002188	0.000045	0.002529
15	Ni_Zn	0.003793	0.000078	NaN
16	h2o_distilled	0.000024	0.000002	-0.000009
17	stainless_steel	0.008420	0.134091	NaN
18	soda_lime_glass	0.000033	0.000004	-0.000011

Figure 10: A table with our results

However, there were a couple of surprises! I was shocked at the lack of literature data for comparison. Most of the data that I *was* able to find was not available in SI units. The conversion from χ_{molar} in cgs to volume χ_m in SI must be done in 2 steps. First, you convert to volume χ_m in cgs and second, you multiply by the conversion factor of 4π. These units are very picky and I still don't fully understand why.

Also, it seems as though something went a bit wrong when we prepared our own samples! While the majority of our data falls within 3 σ of the literature values, we did not even get the correct sign for Water or Soda Lime Glass. Maybe we should have been a bit more suspicious of the bottle labeled "distilled water."

3 Questions

3.1 Units Of χ_m

There are a few common units for χ_m because there are a few different flavors of susceptibility. In class we discussed that Volume χ_m is actually unitless. This is true in both cgs and SI - however, there *is still a conversion factor between* *them*. I find that pretty strange, but I learned to stop questioning units when I started measuring the mass of the sun in kilometers.

You might also encounter a molar χ_m. In CGS this has units of $\frac{cm^3}{mol}$ but Brian told us that in SI, molar χ_m has units of $\frac{kg}{mol}$. That's pretty neat, but Wikipedia disagrees with Brian. Wikipedia says that molar χ_m has units of $\frac{m^3}{mol}$ in SI.

The last type of χ_m that Brian mentioned is mass χ_m. In SI this has units of $\frac{m^3}{kg}$ while in cgs it is $\frac{cm^3}{g}$.

3.2 Compare Results to Literature Values

It's tough to compare our results to literature values, because quite a few of the literature values don't exist. And, when they do exist they're in the wrong units. So, in order to do any comparison I had to start by collecting as many literature values for χ_m as I could find. I converted the cgs values of χ_{mol} into volume χ_m in SI like this:

$$\chi_{sivol} = 4\pi \frac{\chi_{cgsmol}}{M_{cgs}/\rho_{cgs}}$$

Where M is the molar mass. If we plot this with the literature values along the x axis and measure values along the y axis, we would expect the line y = x to intersect most of them. I went one step further and ran a linear fit on our data points using the same curve_fit method that I describe in section 2.1. Plotting the results, we see that, overall, our data agrees with the literature:

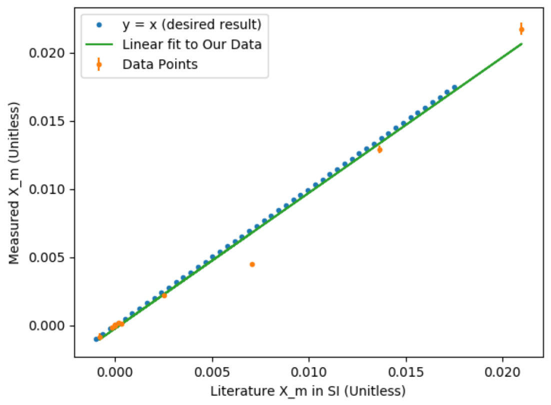


Figure 11: Plotting accepted values for χ_m against our measurements

3.3 Levitation!!

Brian gave this one away. I'm going to go out on a limb here and say that pyrolitic graphite might work, just maybe. Here's what it looks like:

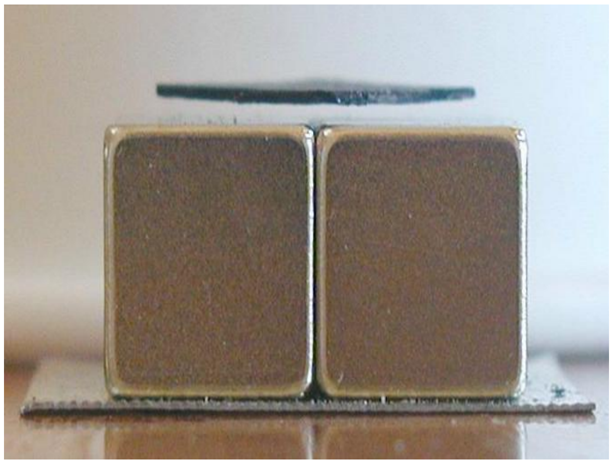


Figure 12: Photo credits to the kind folks over at: http://sci-toys.com

Sci-toys has a really great explanation of the phenomenon. It's worth reading if you like science. Here's an excerpt that will (hopefully) get you excited enough to open up a browser:

> "We can do that by using four magnets. The poles of the magnets push on the diamagnetic material more strongly than other parts of the magnet. With four magnets, the four edges of the square of pyrolytic graphite will be pushed away from the four poles."
> - sci-toys.com

4 Conclusion and Sources of Error

I must say, I'm pretty impressed with our results. I fully expected them to wildly disagree with the accepted values of χ_m. During the experiment, we noticed that the instruments were incredibly sensitive. Things like leaning on the table, or rotating the balance made noticeable impacts upon results. We also noticed that the scale's calibration had a tendency to walk. And, paralax made it quite difficult to ensure that the bottom of our samples was precisely in the middle of the magnetic field.

The only significant source of error seemed to occur during the preparation of our own samples. It's possible that the vials were not adequately cleaned. It's even possible that the samples we prepared were not quite exactly what we thought they were. Lastly, it's possible that the literature values for χ_m are wrong.

If I were to do this experiment again, I would be more careful to monitor the walking of the scale's calibration and more careful while preparing my own samples.