On the Use of a Michelson Interferometer to Determine Indices of Refraction

John Grasel

5/08/2011

Abstract

The index of refraction of helium gas was measured by recording the number of fringes traversing the output of a Michelson interferometer as the helium pressure in an optical cell increased. The LabView code was extended to use a remote clicker to more accurately identify the pressure in the chamber. Using an algorithm to compute the number of fringes between data points, the index of refraction of helium was determined to be $1 + (36.01 \pm 0.06) \times 10^{-6}$. The relation between pressure and time was determined to correspond closely to that of compressible isothermal flow. Using the additional pressure data and a new Fourier transform analysis, more accurate results for the index of refraction of helium, $1 + (35.98 \pm 0.05) \times 10^{-6}$ were determined.
1 Experimental Setup and Theory

A Michelson interferometer generates a circular interference pattern from an incident laser source. The pattern’s bulls-eye shape results from constructive interference from light that traveled an integral number of wavelengths longer or shorter in the translatable arm than the fixed arm, and the destructive from light that traveled a whole number and a half wavelengths longer or shorter in the translatable arm than in the fixed. From counting the fringe signal’s passing minima and maxima, small changes in the optical path length can be measured.

The experimental setup is shown in Figure 1. A laser of known wavelength is used as our light source. A hollow cell of known length is inserted in one of the interferometer’s paths. This cell is attached to two valves and a pump that allow gas to be added or removed. Moving one of the mirrors a distance $D$ from equal path length causes a number of fringes $m$ to pass:

$$m = \frac{2D}{\lambda}$$

where $\lambda$ is the wavelength of the laser and the factor of 2 is because light transits the distance twice. While neither mirror is physically moved, in one arm, the beam is forced through a cell with length $d$ with index of refraction $n$, yielding:

$$m = \frac{2d(n - 1)}{\lambda}$$

Figure 1: **Setup** - A vacuum cell is placed along one path of a Michelson interferometer. Changing the pressure of gas in the cell alters path length, generating interference fringes that are recorded by the photodetector and analog to digital converter.
To change the index of refraction, we change the pressure. The index of refraction $n$ is dependent on pressure $P$, so the formula becomes

$$m = \frac{2d \left(n[P] - 1\right)}{\lambda}$$

For shorthand, $n$ will be used instead of $n[P]$. Values for the index of refraction are quoted at atmospheric pressure $P_0$. The index of refraction minus one is proportional to the pressure, so

$$n - 1 = \alpha \frac{P}{P_0}$$

where $\alpha$ is the proportionality constant. The index of refraction is defined to be one in a vacuum, where $P = 0$, which is satisfied by our formula. The index of refraction at atmospheric pressure is the quantity of interest:

$$n[P_0] - 1 = \alpha$$

Substituting our definition of $n$ from Equation (1),

$$m = \frac{\alpha P \cdot 2d}{\lambda P_0}$$

Substituting in $\alpha$,

$$m = \frac{(n - 1)P \cdot 2d}{\lambda P_0}$$

Now, all variables are fixed except $P$ and $m$, so changing one changes the other.

$$\Delta m = \frac{(n - 1)\Delta P \cdot 2d}{\lambda P_0}$$

Rearranging terms,

$$n - 1 = \frac{\Delta m \lambda P_0}{\Delta P \cdot 2d}$$

The objective of the experiment is find the number of fringes $m$ at various pressures $P$. A plot of $m$ vs. $P$ has a slope of $\Delta m/\Delta P$, which by Equation (2) is proportional to the index of refraction. The counting of the fringes is accomplished by centering the photodiode on the circular fringes and digitizing the intensity. Interrupting the laser with an index card at intervals of equal pressure tags the fringes with marks equally spaced in pressure. For this paper, these intervals of equal pressure are referred to as “isobaric intervals”\(^1\). The fringe intensity sampled at isobaric intervals is referred to as “isobaric data”. Counting the cumulative number of fringes that have passed from the first isobaric data point to each of the rest yields values for $m$ at different $P$.

\(^1\)Not to be confused with a system at constant pressure!
2 The Index of Refraction of Helium

2.1 Experiment Adjustments

The largest change in the procedure is ensuring that helium is the only gas within the chamber. Since its index of refraction (minus one) is one tenth that of air, a small amount of air can tamper with results. The vacuum pump can lower the pressure in the chamber to around 8 mmHg, and the system pressure can be safely raised to 800 mmHg. Thus, one "flush" of the system, which consists of evacuating as much of the air as possible and filling it with helium, removes roughly $1 - \frac{8}{800} = 99\%$ of the air, assuming the helium in the tank is pure. Repeating this process 5 times results in nearly pure helium. In addition, the system was left at a near-vacuum state for ten minutes, and there was no noticeable change in pressure, so leakage is not a concern.

The given procedure is effective for measuring large indices of refraction, like that of air, but it has some flaws. Firstly, in order to mark the chamber pressure, a mark is made by interrupting the laser beam, but this interrupts valid data as well. In addition, a human can only determine the position of an interruption within a fringe period to 1 part in 16. This is insufficient to measure the refractive index of helium. In addition, the computer can use precise algorithms to determine fractional wavelength, and they have the advantage of being consistent; i.e, if it consistently overestimates the fractional wavelength, that contributes only to the offset of the $m$ vs. $P$ graph, not its slope. Computers can record a click with better resolution than an experimentalist can wave an index card, and it requires less effort on the experimentalist. A factor of 10 to 20 more pressure data points can be acquired. The computer can process the data rapidly and generate plots to ensure that the algorithm is working properly. These adjustments require altering the existing LabView code and writing processing software in Python.

2.1.1 LabView Modifications

The LabView code was modified to save a list of data points when a key- board or slide advancer key is pressed. Since the sample rate is 100 samples per second, the time in seconds of any press is equal to its data point index divided by 100. The average click is represented by 5-8 samples, corresponding to 0.05-0.08 seconds. This is much more responsive than the previous method. The clicker used does not register a small fraction of the clicks (roughly 1 in 50), but this is corrected for in the signal processing. It is not known whether this is a hardware problem in the clicker, an error in the wireless transmission, or a software issue.
2.1.2 Python Digital Signal Processing

The processing code was written in Python; the code can be found on Pages 50 through 57. After parsing the data files, the average of consecutive press points is taken to determine when keys were pressed. An algorithm checks to see if any clicks appear to be missing, using the fact that the time between clicks is roughly linear. If a gap is found, the algorithm will guess the skipped click occurred halfway between the clicks on either side. This method will fix any single missing point, but will fail when multiple points are skipped in a row.

As Figure 2 shows, the raw data is too noisy to directly compute the extrema or $m$. The data is smoothed by convolving it with a Hanning window. The window’s time length is proportional to the amount of smoothing. A convolution method has the advantage of not shifting the smoothed data in time. Before finding the extrema, heavy smoothing (8 seconds time length) is applied to the raw data (see Figure 3). Insufficient smoothing can result in false extrema locations, but heavy smoothing makes the extrema locations more accurate with no negative consequences. After all, smoothing is simply a low-pass filter, and the fringe period is far too large to be negatively affected. A simple search algorithm determines the locations of the minima and maxima on the smoothed signal.

Next, the number of fringes between isobaric data points is determined; this array is summed cumulatively to calculate the number of extrema for each pressure. The result is shown in Figure 5. To iron out some of the photodetector fluctuations on a small time scale, very light filtering (0.35 seconds time length) is applied (see Figure 4).

There are two methods of determining the number of fringes between extrema. One, called X, works in the horizontal direction; the other, Y, works in the vertical direction. The X method computes the fraction $F$ of the point $P$’s horizontal location in $x$ (time) compared to the horizontal location of the neighboring extrema, $E_n$ and $E_{n+1}$:

$$F_x = \frac{P.x - E_n.x}{2E_{n+1}.x - E_n.x}$$

For the X method, since only the horizontal locations are used, it does not matter whether $E_n$ and $E_{n+1}$ are a minima and a maxima, or a maxima and a minima.
Figure 2: **Raw Data** - The raw fringe intensity recorded by the detector over time. The noise at the beginning and the end is the result of opening and closing the intake valve, respectfully. The data is too rough to directly find the minima and maxima. In addition, the data is too noisy to use it to find $m$. 

Figure 3: **8-Second Smoothing Data** - Data after applying heavy smoothing. This is only used to accurately find the maxima and minima, not to determine the index of refraction. With data this smooth, the extrema-finding algorithm becomes simple. Care was taken to ensure that the smoothing did not delay the data in time. 

Figure 4: **0.35-Second Smoothing Data** - Data after applying slight smoothing to raw data. This makes the data less noisy, and the resulting values for $m$ more linear when plotted against pressure.
The Y method calculates the horizontal position by using vertical direction $y$ (voltage) with some simple trigonometry:

$$F_y = \frac{1}{4} \pm \frac{3}{2\pi} \arcsin\left(\frac{2P.y - E_n.y - E_{n+1}.y}{2(E_{n+1}.y - E_n.y)}\right)$$

The plus sign is chosen when $E_n$ is a minima and $E_{n+1}$ is a maxima, and the minus sign when the opposite is true.

The X method is inaccurate when the fringe period is changing rapidly, while the Y method is not affected by the fringe period. The Y method depends on the maximum intensity being constant over time, and it is less accurate near extrema due to the inverse sine function, but the X method does not have these limitations. From experimentation, the Y method works best with air, when there are lots of fringes and thus a plethora of vertical data, and the changing fringe period leaves the X method inaccurate. The X method works better for helium, though, because there are fewer fringes. Both methods have weaknesses, but the mean difference between the X and Y method:

$$\frac{1}{N-1} \sum_{n=1}^{N-1} |F_x - F_y|$$

is a very conservative estimator of the algorithm’s effectiveness; results show it is 1 part in 30, roughly twice as accurate as a human. While these methods may be better than manual estimation, but as they are still non-ideal, there is still benefit from verifying their effectiveness or circumventing their use.

2.2 Data and Results

The charts of pressure versus fringes are shown in Figures 8, 9, 10, 11, 12, 13, 14, and 15. Each chart is the result of five evacuations and fills followed

![Figure 5: Final Result - The 0.35-second smoothed data overlayed with identified extrema (red) and isobaric data points (green). In this case, the first and last extrema locations are inaccurate, so X-method calculations would only be valid between indices 2,000 and 13,500.](image)
Table 1: The index of refraction of helium was determined to be 1.00003601 ± 0.00000006. The Len column shows how long each data run took, in seconds. Most runs were taken over the course of 5 minutes, with a few taking closer to 10 minutes. In one run, at 14:42, the cell was pressurized much more rapidly, in less than 3 minutes, and its calculated index of refraction is significantly different from the others. This difference is likely attributable to inaccuracies introduced when rapidly taking data. Values for the index of refraction were calculated with and without this outlier, but it is excluded from the final result. All indices of refraction are expressed at atmospheric pressure at 0°C, not the temperature at which they were measured.

<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
<th>Temp (°C)</th>
<th>Len (s)</th>
<th>(n-1)×10^{-6}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13:30</td>
<td>21.2</td>
<td>290</td>
<td>36.13</td>
</tr>
<tr>
<td></td>
<td>16:36</td>
<td>21.7</td>
<td>701</td>
<td>36.19</td>
</tr>
<tr>
<td>2</td>
<td>14:11</td>
<td>20.0</td>
<td>324</td>
<td>35.85</td>
</tr>
<tr>
<td></td>
<td>14:19</td>
<td>20.6</td>
<td>314</td>
<td>35.78</td>
</tr>
<tr>
<td></td>
<td>14:28</td>
<td>21.2</td>
<td>352</td>
<td>36.05</td>
</tr>
<tr>
<td></td>
<td>14:42</td>
<td>21.4</td>
<td>157</td>
<td>35.11</td>
</tr>
<tr>
<td></td>
<td>14:55</td>
<td>21.5</td>
<td>641</td>
<td>35.98</td>
</tr>
<tr>
<td></td>
<td>15:06</td>
<td>21.5</td>
<td>506</td>
<td>36.06</td>
</tr>
</tbody>
</table>

Average - All 35.89 ± 0.12
Average - No Outlier 36.01 ± 0.06

by one slow fill where data is recorded. The error bars on the data are not uncertainties calculated from a standard deviation; unfortunately, it is difficult to combine data from different runs since the temperature changes. The error bars are the result of Equation (3); they were added to judge the effectiveness of X and Y methods of determining m. As such, the error bars, the reduced $\chi^2$ values, and the uncertainties on the fitting parameters have no statistical meaning. In general, the linear fits matched the data well, with one exception: the short data run. The length of the data run is determined by the position of the needle valve that lets gas in. An open needle resulted in rapid pressurization of the cell, noisier data (see Figure 13), and an erroneous value for the index of refraction of helium. This data run is excluded from the calculated index of refraction of helium. At the other extreme, the longest data run, in Figure 9, shows patterned residuals that may be the result of thermal drift over time or other longer-term effects.

The results are shown in Table 1. The uncertainty in the index of refraction comes from the standard error from the seven data runs. The data from the first day was not statistically different from the data from the second day,
so the results were combined. There is certainly no consensus in previous results, as seen in Figure 6. In fact, there seem to be two distinct camps of results, one near 34.75, and the other near 36.0. This new data is the most precise value seen, possibly due to the new methods of data analysis and the acquisition of seven data runs instead of the standard five.

3 Investigating the Pressurization of the Cell

Collecting information via computer yielded extra information about the pressure inside of the cell over time. With hundreds of pressure data points spaced at intervals of 5 mmHg, it becomes possible to see whether the actual data matches theory. If so, a curve of best fit of pressure over time can be computed, which potentially averages out the random error inherent in clicking a button at equal-pressure intervals.

3.1 Theory of Pressure Drop Along a Pipe

Here, an attempt is made to derive the theoretical pressure in the cell over time. In order to simplify the theory, a few assumptions must be made. The flow is isothermal because the experiment is not insulated and is exposed to the ambient environment, and no mechanical work is done on or by the gas. The gas is ideal, which is a good assumption for air at atmospheric pressure, and it is even better for lighter gases like helium and gases at low pressure. The friction factor is constant, and the pipe is straight and horizontal. Then,

Figure 6: There is little consensus within Harvey Mudd regarding the index of refraction of helium. However, this new data, along with that of two other students, agrees with literature values near 36.
the equation for compressible isothermal flow is

\[ w^2 = \frac{D\rho_1 A^2}{fL} \frac{p_1^2 - p_2^2}{p_1} \]

where \( w \) is the average mass flow rate, \( p_1 \) and \( p_2 \) are the pressures at the beginning and end of the pipe, \( \rho_1 \) is the gas density inside the gas reservoir, \( f \) is the friction factor, \( L \) is the pipe length, \( D \) is the pipe diameter, and \( A \) is the cross-sectional area\(^2\). By the ideal gas law,

\[ PV = nRT \implies \frac{dP}{dt} \propto \frac{dn}{dt} \]

and the mass flow rate \( w \) divided by the molecular mass \( m \) is the molecule flow rate, or

\[ \frac{w}{m} = n \implies \frac{dw}{dt} \propto \frac{dn}{dt} \]

Combining constants,

\[ w^2 \propto \frac{p_1^2 - p_2^2}{p_1} \implies w \propto \sqrt{\frac{p_1^2 - p_2^2}{p_1}} \]

so

\[ \frac{dP}{dt} \propto \sqrt{\frac{p_1^2 - p_2^2}{p_1}} \]

giving us our differential equation, with \( P_0 \) as the outside pressure and \( P \) as the cell pressure,

\[ \frac{dP}{dt} = C_1 \sqrt{\frac{P_0^2 - P^2}{p_1}} \]

Solving this differential equation,

\[ P = P_0 \sin \left( \frac{C_1 t + C_2}{\sqrt{P_0}} \right) \quad (4) \]

At first glance, this periodic solution seems nonsensical, since thermodynamics states that the system will never move away from equilibrium. However, if the solution is plugged back into the differential equation,

\[ \cos \left( \frac{C_1 t + C_2}{\sqrt{P_0}} \right) = \sqrt{\cos^2 \left( \frac{C_1 t + C_2}{\sqrt{P_0}} \right)} \]

\(^2\text{From \url{http://www.pipeflowcalculations.com/pipeflowtheory/pressure-drop-equation-in-isothermal-flow.php}}\)
which occurs only when
\[
\frac{C_1 t + C_2}{\sqrt{P_0}} < \frac{\pi}{2}
\]
Solving for \( t \) and plugging that value back into the solution, it is seen that the solution only applies when \( 0 \leq P \leq P_0 \). Since this is the region of interest, the solution holds.

3.2 Results

The experiment pressure data was fit to the theoretical prediction; firstly, the data taken during the second experiment, when the index of refraction of air was determined, was used. The plots for air data are shown in Figures 16, 17, 18, 19, and 20. For all but one graph, the model is a very good fit. The pressure data does not have associated uncertainty, since each data point is a single measurement, so the reduced \( \chi^2 \) values are meaningless, but the residuals are small, and a value of \( P_0 = 750 \text{ mmHg} \) is obtained, which is close to atmospheric pressure. Every plot has identically-patterned residuals, but there aren’t enough data points to draw serious conclusions.

Fortunately, the helium pressure data has an order of magnitude more points than its air counterpart. The plots are shown in Figures 21, 22, 23, 24, 25, and 26. The data fits the data incredibly well, with residuals under 5 mmHg in nearly all cases; this is even more impressive because over a hundred data points were taken, and the pressure dial’s tick resolution is 5 mmHg. One oddity is that the calculated \( P_0 \) is around 900 mmHg, but theoretically it should be around 1250 mmHg\(^3\). This could be a result of the fit not working as well at high pressures, or perhaps there is an issue with the helium regulator.

The most interesting feature of all but one of these graphs is they share the same patterned residuals as the air graphs. The residual pattern looks like 80% of a sinusoid. The fact that it is present on nearly every graph suggests that there is either an issue with the model, or there is an issue with the experiment. There were no blatant assumptions in the model, and the data conforms so well to the fit, that it is unlikely that there is an issue with the model. The residuals are so similar that they’re unlikely to be caused by thermal fluctuations. After examining several causes, it has been hypothesized that the residuals are an artifact of a tall experimenter looking down on the setup, an effect dubbed “perspective error”. At low pressure, when the needle is facing southwest, a tall experimenter reads a slightly lower value than is actually there. The effect reverses at medium pressure, about 400 mmHg, where the needle is vertical. The needle continues clockwise until it’s vertically down near 800 mmHg. The geometry of the setup suggests

\(^3\)1250 mmHg \approx 1 \text{ atmosphere} + 10 \text{ psi}
that an experimentalist’s constant head position looking at a needle spinning around a dial would yield residual errors that were sinusoidal and of a magnitude comparable to what is seen in the results. The final mystery is why Figure 24 on Page 34 has completely flat residuals; this was the data point excluded from the value of the index of refraction of helium because the data was taken so quickly. It’s possible that the needle was spinning so rapidly, the experimenter leaned in close to see when it was passing pressure points, which could eliminate the perspective error. Additional testing is necessary to verify this hypothesis.

4 Fourier Data Analysis

It has been demonstrated that there is a way to accurately find pressure over time from experimental data. This data can be used to obtain more accurate values for the index of refraction of air.

4.1 Pressure Domain Theory

4.1.1 Non-Mathematical Overview

Figure 7 visualizes the non-mathematical concepts of the old method, which will hereafter be referenced as time-domain analysis, and the new technique, which is called pressure-domain analysis. Recall that the time-domain method involved marking a small number (around 150, $\Delta P = 5$ mmHg) of equally-spaced-in-pressure points using an automated clicker, or worse, interrupting the laser beam to take an even smaller number (around 15, $\Delta P = 50$ mmHg) points. These points are used to sample the time-domain fringe intensity signal, keeping tens or hundreds of fringe amplitudes out of tens of thousands recorded by the analog to digital converter. The experimenter can manually estimate the fractional position of each point in the fringe, or the computer can do so digitally, but neither procedure is accurate nor precise. Recall that the slope of the number of fringes vs. pressure plot yields a result proportional to the index of refraction of air. There are two reasons pressure-domain analysis is more effective: doesn’t throw out over 99% of the fringe data\textsuperscript{4}, and it avoids the fuzzy procedure of determining the number of fringes between isobaric data points outlined in Section 2.1.2 on Page 6.

The pressure-domain method uses the equation relating time to pressure (Equation 4 on Page 11) to fit the pressure data over time. This information is used to sample the fringe intensity linearly in pressure, keeping the huge amount of fringe intensity data recorded and putting it to use - the frequency\textsuperscript{4}

\textsuperscript{4}At best, the sampling keeps hundreds of data points out of tens of thousands
of fringe intensity spaced linearly in pressure is also proportional to the index of refraction. This frequency can be found by fitting the data to a sinusoid, or by taking its Fourier transform.

![Diagram](image)

**Figure 7: Technique** - Originally, an analog to digital converter samples the fringe intensity (upper left, blue line) linearly in time with a large number of data points (over 9000), and the experimenter records a small number (15 normally, or 150 if the automated clicker is used) of points equally spaced in pressure (red dots). Then, the position of each red dot along the fringe is estimated, and the result is plotted on the upper right. The slope of this line, as shown in Equation 2 on Page 4, is proportional to the index of refraction. Alternatively, the equation correlating pressure and time (Equation 4 on Page 11) could be fit to the pressure data and used to sample the fringe intensity linearly in pressure, shown in the lower middle figure. The frequency of this sinusoid, which has units of fringes per pressure unit, is a factor of $2\pi$ different from the slope of the line in the upper right-hand plot. The upper-middle plot is the version of lower-middle plot, but with a small number of points as opposed to a large number. As opposed to calculating the obscure fringe fraction in the original method, the new method involves simply calculating the frequency of a sine wave. This can be done through data fitting or a Fourier transform.
4.1.2 Mathematical Analysis

The fringe intensity of a non-ideal Michelson interferometer is given by the equation

\[ I = I_{\text{max}} \cos^2 \left( \frac{2\pi}{\lambda} (L_1 - L_2) + \phi \right) + I_{\text{min}} \]

where \( \lambda \) is the laser wavelength, \( L_1 \) and \( L_2 \) are the fixed and movable path lengths, and \( \phi \) a phase term added because the equal-path length position can be anywhere in the fringe. Remember that Equation 1 on Page 4 stated

\[ L_1 - L_2 = 2(n - 1) \frac{P}{P_0} \]

so the intensity relation, now a function of pressure, becomes

\[ I[P] = I_{\text{max}} \cos^2 \left( \frac{4\pi d}{\lambda P_0} (n - 1) P + \phi \right) + I_{\text{min}} \]  \hspace{1cm} (5)

If the data is fit to a curve,

\[ I[P] = I_{\text{max}} \cos^2 (CP + \phi) + I_{\text{min}} \]

where \( C \) is a constant, then the index of refraction can be found:

\[ n - 1 = C \frac{P_0 \lambda}{4\pi d} \]  \hspace{1cm} (6)

Note that \( C \) is related to the slope \( \Delta m/\Delta P \) from the linear fit in the old method by the equation:

\[ C = 2\pi \frac{\Delta m}{\Delta P} \]  \hspace{1cm} (7)

Using a trigonometry identity, \( \cos x^2 = (1 + \cos x)/2 \), we can also fit to the curve

\[ I[P] = I_{\text{max}} \frac{\cos(2CP + 2\phi) + 1}{2} + I_{\text{min}} \]  \hspace{1cm} (8)

4.1.3 Fourier Transforms in Pressure

Once the fringe intensity is resampled to be equally spaced in pressure, the sinusoid’s period must be found. This can be found with a Fourier transform or by fitting the data to Equations 5 or 8. In order to determine which to use, Fourier transforms in pressure space must be investigated. Any function \( f[x] \) defined for \(-L < x < L\) can be written as

\[ f[x] = a_0 + 2 \sum_{n=1}^{\infty} a_n \cos (nx) + 2 \sum_{n=1}^{\infty} b_n \sin (nx) \]
where
\[ a_n = \frac{1}{L} \int_{-L}^{L} f(x) \cos (nx) \, dx \]
\[ b_n = \frac{1}{L} \int_{-L}^{L} f(x) \sin (nx) \, dx \]

If \( c_n \) is defined as \( a_n - b_n \) when \( n > 0 \) and \( c_{-n} \) when \( n < 0 \), then the series can be rewritten as
\[
f(x) = a_0 + 2 \sum_{n=1}^{\infty} a_n \left( e^{inx} + e^{-inx} \right) + \frac{b_n}{i} (e^{inx} - e^{-inx})
\]
\[
= \sum_{n=1}^{\infty} (a_n - b_n) e^{inx} + (a_n + b_n) e^{-inx}
\]
\[
= \sum_{n=-\infty}^{\infty} c_n e^{inx}
\]
(9)

Since \( I[P] \) is also a function, it can be expressed as a Fourier series:
\[
I[P] = \sum_{n=-\infty}^{\infty} c_n e^{inP}
\]

Let the intensity data have \( N \) terms at pressures from \( P_{min} \) to \( P_{max} \), a range of \( P_{range} = P_{max} - P_{min} \). The data is equally spaced at pressure intervals of \( P_{range}/N \). With a finite number of terms \( N \) in the pressure data, a close approximation is made instead. First it is noted that \( I \) can be thought of as a function of pressure, or as “the \( n \)’th element of the intensity array” \( I_n \), so
\[
I\left[ P_{min} + \frac{P_{range}}{N} n \right] = I_n
\]

Thus, the approximations become
\[
C_n \approx \sum_{k=0}^{N-1} I_k e^{ink}
\]
\[
I[P] \approx \frac{1}{N} \sum_{n=-(N-1)/2}^{(N-1)/2} C_n e^{2\pi i n P / P_{range}}
\]

where the \( 1/N \) factor in the \( c_n \)s was moved from the \( C_n \)s to \( I[P] \). The critical achievement is that the top equation is simply the discrete Fourier transform
of the intensity, which is easily computed! Now, if this is rewritten in the form of Equation (8),

$$I[P] = \frac{1}{N} \left( C_0 + \cmax e^{2\pi i n_{\text{max}} P/P_{\text{range}}} + C_{-n_{\text{max}}} e^{-2\pi i n_{\text{max}} P/P_{\text{range}}} \right)$$

(10)

where \( n_{\text{max}} \) is the index of the peak intensity in the Fourier spectra. We can rewrite Equation (8) as

$$I[P] = \left( I_{\text{min}} + \frac{I_{\text{max}}}{2} \right) + \left( \frac{I_{\text{max}}}{4} e^{2i\phi} e^{2iCP} \right) + \left( \frac{I_{\text{max}}}{4} e^{-2i\phi} e^{-2iCP} \right)$$

Comparing this with Equation (10), the Fourier coefficients are related to physical parameters:

$$\frac{|C_{n_{\text{max}}}|}{N} = \frac{I_{\text{max}}}{4}$$

$$\frac{C_0}{N} = I_{\text{min}} + \frac{I_{\text{max}}}{2}$$

$$C = \frac{\pi n_{\text{max}}}{P_{\text{range}}}$$

Solving the equations for the variables of interest,

$$I_{\text{max}} = 4 \frac{|C_{n_{\text{max}}}|}{N}$$

(11)

$$I_{\text{min}} = \frac{C_0}{N} - \frac{I_{\text{max}}}{2} = \frac{C_0 - 2|C_{n_{\text{max}}}|}{N}$$

(12)

$$C = \frac{\pi n_{\text{max}}}{P_{\text{range}}}$$

(13)

As a recap, taking the Fourier transform of intensity data equally spaced in pressure yields a coefficient vector. The index of the maximum element, \( n_{\text{max}} \) and its value \( C_{n_{\text{max}}} \) are found, and from these values, all variables of interest can be derived, including \( C \), which is proportional to the index of refraction (Equation 6).

4.1.4 Choosing Sinusoidal Fitting or Fourier Transforms

Now, two different ways are available to calculate the value \( C \) from intensity data - sinusoidal fitting, or the Fourier transform. The Fourier transform method is limited by the resolution of the peak specified \( n_{\text{max}} \). The peak index is just the closest integer to the “true” value, so the uncertainty on \( n_{\text{max}} \) is 1/2. Thus the uncertainty on \( C \) is given by

$$\Delta C = \frac{\pi}{2P_{\text{range}}}$$

(14)
The experiment uncertainty can be defined as

\[
\text{Uncertainty} = \frac{\Delta(n - 1)}{n - 1} = \frac{\Delta C}{C} = \frac{\pi}{2\pi \Delta P} = \frac{1}{4 \Delta P} = \frac{1}{4m}
\]

where Equation 7 was used for \(C\). In the last step, multiplying a pressure range times a slope representing the change in the number of fringes per unit pressure results in the total number of fringes traversed \(m\). Thus, the error is inversely proportional to the number of fringes! For helium, a maximum of 5.5 fringes can be acquired over a safe pressure range, so the uncertainty in the index of refraction is around 12%, an unacceptably high value. However, for air, a maximum of 45 fringes can be acquired, so the uncertainty is only 0.6%, a much more acceptable value.

Fortunately, the Fourier transform’s weakness is the sinusoidal fit’s strength: when there are only a few fringes, a sinusoidal fit can accurately calculate the frequency; however, when there are many fringes, the fit is unable to adapt to minuscule changes in frequency and amplitude, and these can compound until the fit no longer agrees with the data. In summary, for gases with refractive indices less than or close to that of helium, sinusoidal fitting should be used. For gases with indices of refraction greater than or near air’s index of refraction, Fourier transforms are more effective. For the region of gases between helium and air, either method will work.

### 4.2 Coding the Resampler

The algorithm for resampling the data linearly in pressure is relatively straightforward, and the code is attached in the back on Page 59. As before, pressures are bunched and the clicks where reliable data starts and ends are noted. Two functions are defined: \(P[x]\), which takes a data point index and turns it into a pressure, and \(X[p]\), which does the inverse. \(P\) is used to determine the pressures where good data starts and ends. The intensity data, which starts linearly in time, is interpolated linearly. A linear space of pressure points is created from the start of the accurate pressure data to its end, and each pressure is applied to \(X\), which returns its corresponding point. All of these points sample the interpolating function, creating a new function of fringe intensities that is sampled linearly in pressure.

### 4.3 Helium Results

Table 2 shows the results from fitting fringe amplitude data that’s equally spaced in pressure to a sinusoid. The graphs are shown in Figures 27, 28, 29, 30, 31, and 32. The calculated index of refraction is 35.98 ± 0.05, calculated with the mean and standard error of the six trials. The mean very similar to
Table 2: The index of refraction of helium, 35.98 ± 0.05, is nearly identical to the previously attained value of 36.01 ± 0.06 once the outlier is excluded. The uncertainty is slightly smaller, however. The sinusoidal fitting process was not carried out for the data taken on the first day.

<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
<th>Len (s)</th>
<th>(n-1)×10⁻⁶</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>14:11</td>
<td>324</td>
<td>35.95</td>
</tr>
<tr>
<td></td>
<td>14:19</td>
<td>314</td>
<td>35.78</td>
</tr>
<tr>
<td></td>
<td>14:28</td>
<td>352</td>
<td>36.02</td>
</tr>
<tr>
<td></td>
<td>14:42</td>
<td>157</td>
<td>35.01</td>
</tr>
<tr>
<td></td>
<td>14:55</td>
<td>641</td>
<td>36.06</td>
</tr>
<tr>
<td></td>
<td>15:06</td>
<td>506</td>
<td>36.07</td>
</tr>
</tbody>
</table>

Average - Day 2: 35.82 ± 0.17
Average - No Outlier: 35.98 ± 0.05

the mean calculated with the original methodology, and an uncertainty 20% less. The fitting models agree well with the data, but once again, with no uncertainties in the data, the $\chi^2$ values are meaningless. The data seemed to fit the worst near the peaks, where small fluctuations in the data were apparent. The residuals appear mostly random, but there appears to be a small oscillatory component. This new method lowers the propagated uncertainty on each individual run’s implied index of refraction by 20%, but the propagated uncertainty is still half of the uncertainty calculated by the standard error of the five included data runs.

While it could be argued that a 20% reduction in uncertainty wasn’t worth the trouble, using a second method to obtain similar results is useful. It validates both routines, which is no small accomplishment. For the curve fitting technique, it confirms that fitting intensity over pressure is a valid method of finding the index of refraction. For the original technique, it suggests that the determined algorithm of locating a point’s fraction within a fringe is very effective. In order to calculate a more precise result, temperature, which varied by as much a few tenths of a degree Celsius, might have to be controlled.

### 4.4 Air Results

#### 4.4.1 Sinusoidal Fitting for Air

The sinusoidal data fitting technique was tested against previous data gathered during Experiment 2 with air as the gas. This data has several problems;
Table 3: The index of refraction of air is fairly different from the value of tabulated value, 2.77.

<table>
<thead>
<tr>
<th>Time</th>
<th>Len (s)</th>
<th>((n-1)\times10^{-4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:49</td>
<td>844</td>
<td>2.86</td>
</tr>
<tr>
<td>12:13</td>
<td>431</td>
<td>2.80</td>
</tr>
<tr>
<td>12:52</td>
<td>550</td>
<td>2.81</td>
</tr>
<tr>
<td>13:31</td>
<td>616</td>
<td>2.80</td>
</tr>
<tr>
<td>13:39</td>
<td>251</td>
<td>2.75</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>2.80 ± 0.02</strong></td>
</tr>
</tbody>
</table>

firstly, there were only a dozen or so pressure data points taken, so the pressure over time curve is not as reliable; secondly, with so many fringes, it’s difficult to fit them all. The graphs are shown in Figures 33, 34, 35, 36, and 37. As expected, the fits are abysmal. Only Figure 37 had a good fit; every other graph has a changing fringe frequency, which causes the sinusoidal fit to fail. Figure 37 was the shortest data run - over longer periods, small heat changes can change the frequency enough to ruin the fit. One of the advantages of the Fourier transform method is that this problem is avoided. Table 3 shows the data. The index of refraction of air, \(2.80 \pm 0.02\), is different from the tabulated value, 2.77, but not by a huge margin.

### 4.4.2 Fourier Transform Code

The Fourier transform Python code is attached on Page 61. The code loads the resampled pressure-domain data from the file. The signal is centered around zero to avoid generating any DC offset. An optional Blackman-Harris window is applied to the data; this reduces spectral leakage and makes the existing frequencies stand out. Unfortunately, these two functions change the fringes’ amplitude, so it becomes impossible to recover the intensity’s minimum and maximum, but they make it easier to find \(C\), the variable we care about most. The Fourier transform of the data is taken, a spectrum is output to a file, and the index of the maximum element \(n_{\text{max}}\) is found. Then Equations 11, 12, and 13 are implemented and the result is returned.

There are two ways to recover the index with the highest Fourier amplitude. The first is by simply taking the maximum, which is discussed above:

\[
    n_{\text{max}} = \text{maxIndex}[f]
\]

if \(f\) is the Fourier transform of the intensity. However, the uncertainty of this method is given by Equation 14, since the maximum index has to be
Table 4: The index of refraction of air computed using the Fourier transform method, as \((n - 1) \times 10^{-4}\). Using the weighted average method greatly lowers the deviation between runs. Both values are significantly different from the tabulated value at 15 °C, 2.766.

<table>
<thead>
<tr>
<th>Time</th>
<th>Len (s)</th>
<th>Max Index</th>
<th>Weighted Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:49</td>
<td>844</td>
<td>2.87 ± 0.04</td>
<td>2.785</td>
</tr>
<tr>
<td>12:13</td>
<td>431</td>
<td>2.82 ± 0.04</td>
<td>2.819</td>
</tr>
<tr>
<td>12:52</td>
<td>550</td>
<td>2.85 ± 0.04</td>
<td>2.788</td>
</tr>
<tr>
<td>13:31</td>
<td>616</td>
<td>2.78 ± 0.04</td>
<td>2.786</td>
</tr>
<tr>
<td>13:39</td>
<td>251</td>
<td>2.73 ± 0.04</td>
<td>2.802</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>2.81 ± 0.03</td>
<td>2.796 ± 0.006</td>
</tr>
</tbody>
</table>

an integer. A more effective algorithm is to find the weighted average of the maximum point and the points left and right of it:

\[
n_{\text{max}} = \frac{(n_{\text{max}} - 1) \times |f_{n_{\text{max}} - 1}| + (n_{\text{max}}) \times |f_{n_{\text{max}}}| + (n_{\text{max}} + 1) \times |f_{n_{\text{max}} + 1}|}{|f_{n_{\text{max}} - 1}| + |f_{n_{\text{max}}}| + |f_{n_{\text{max}} + 1}|}
\]

where \(f_a\) is the a’th element of the Fourier transform. Test code is also included to generate a sample wave and test the calculated \(C\).

4.4.3 Fourier Transform Results for Air

The spectra graphs for air are shown in Figures 38, 39, 40, 41, and 42. Visually, it is clear that the index of refraction is close to literature values, and the 3-point average method appears to be more centered in the peak than the maximum index method. The results are shown in Table 4. Using the weighted average method greatly lowers the deviation between runs because it accounts for nearby points that may be almost as high as the maximum (see Figure 40). Both values are significantly different from the tabulated value at 15 °C, 2.766. This is interesting because a subset of this data in Experiment 2 yielded an answer very close to the actual value. Interestingly, these indices of refraction are close to the those seen using the sinusoidal fitting method, which makes sense because they’re both calculating the frequency of the same resampled intensity data. Given that both of these methods are giving similar and incorrect answers, the problem is likely that the resampled data has an error. The pressure-over-time data for air had only a dozen data points, so the resulting fit might not have been accurate.
Table 5: The index of refraction of helium is computed using the Fourier transform method on Day 2 data, as \((n - 1) \times 10^{-6}\). Using the weighted average method greatly lowers the deviation between runs; as expected, the max index uncertainty, which is calculated not as the standard error but propagated error stemming from the discrete Fourier transform resolution, is too great for results of any significance. However, the weighted average method has a lower uncertainty but gives a different index of refraction from the value of 35.98 ± 0.05 calculated earlier.

<table>
<thead>
<tr>
<th>Time</th>
<th>Len (s)</th>
<th>Max Index</th>
<th>Weighted Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:11</td>
<td>324</td>
<td>33.9 ± 3.4</td>
<td>35.4</td>
</tr>
<tr>
<td>14:19</td>
<td>314</td>
<td>34.0 ± 3.4</td>
<td>35.1</td>
</tr>
<tr>
<td>14:28</td>
<td>352</td>
<td>34.1 ± 3.4</td>
<td>35.5</td>
</tr>
<tr>
<td>14:42</td>
<td>157</td>
<td>33.5 ± 3.4</td>
<td>34.9</td>
</tr>
<tr>
<td>14:55</td>
<td>641</td>
<td>33.8 ± 3.4</td>
<td>35.1</td>
</tr>
<tr>
<td>15:06</td>
<td>506</td>
<td>34.2 ± 3.4</td>
<td>35.5</td>
</tr>
<tr>
<td>Average</td>
<td>34 ± 3</td>
<td>35.24 ± 0.10</td>
<td></td>
</tr>
</tbody>
</table>

4.5 Fourier Transform Results for Helium

Even though it has already been argued that Fourier analysis is probably not great for helium, it was trivial to do it after the code had been written for air. The graphs are shown in Figures 43, 44, 45, 46, 47, and 48. The data is shown in Table 5. As expected, the max index uncertainty is too great for results of any significance. Unlike the weighted average uncertainty, which is the standard error of the results from individual experiments, the max index uncertainty is propagated error stemming from the discrete Fourier transform resolution (see Equation 14). This is because the uncertainty from the theoretical resolution was greater than the standard error. Since both this Fourier method and the sinusoidal fitting method use the same resampled pressure data, the resulting values for the index of refraction should have been closer; however, this can probably be attributed to what had already been hypothesized: Fourier transform analysis should not be used for gases with indices of refraction as low as that of helium.

5 Conclusion

The LabView modifications and the Python fringe estimator developed in Experiment 2 turned out to be very effective in measuring the index of refraction of helium. It also yielded additional pressure and timing data that opened multiple doors for new methods of data analysis. The pressuriza-
tion of the cell over time matched closely to theory, but using that data to determine the indices of refraction had mixed results, with moderate success with helium and the sinusoidal fitting, and less success with the Fourier transform.
6 Graphics

6.1 Helium Fringes vs. Pressure

Figure 8: Day 1 1:30 PM - A linear fit is great for this data. Data run consisted of 5 evacuations and fillings to remove any air from the system, followed by one slow pressurization of the cell while fringe and pressure data were recorded. The data run was a single measurement without uncertainty, so the error bars shown have no statistical meaning; therefore, the uncertainties and the $\chi^2$ values are also meaningless.
Figure 9: Day 1 4:36 PM - This was the longest data run, and its residuals show that other effects such as temperature drifts are affecting the data.

Figure 10: Day 2 2:11 PM - A linear fit is great for this data.
Figure 11: Day 2 2:19 PM - The first dozen data points do not fit the curve well, so the data was masked.

Figure 12: Day 2 2:28 PM - A linear fit is good for this data.
Figure 13: Day 2 2:42 PM - This data was taken in half the time that the others took. The slope is significantly different than the others’ as well. It’s likely that letting the gas in too fast introduced too much noise. This data was excluded in from the calculation of the index of refraction of helium.

Figure 14: Day 2 2:55 PM - A linear fit is alright for this data. This data run was twice as long as the others, so there many be some long-term effects (like thermal fluctuations) in the data.
Figure 15: Day 2 3:06 PM - A linear fit is alright for this data, but the error bars were noticeably smaller for this data set. The error estimator in the Python script isn’t terribly accurate, which is alright since the uncertainty propagation is not critical here.

Coefficient values ± one standard deviation

\[ a = -0.36865 \pm 0.000327 \]
\[ b = 0.0071204 \pm 6.84e-07 \]
\[ \chi^2 = 1.86e+03 \text{ (12.5 / DoF)} \]
\[ P(\geq) = 0 \]
6.2 Air Pressure vs. Time

Figure 16: Air Pressure over Time (from Lab Report 2) - This fit isn’t as good particularly with its patterned residuals. The value for the air pressure is not close to the true value. These graphs also represent single measurements without uncertainty, so the $\chi^2$ value is meaningless.
Figure 17: Air Pressure over Time (from Lab Report 2) - This data fits the theoretical model very well, and gives a reasonable value for atmospheric air pressure.

Figure 18: Air Pressure over Time (from Lab Report 2) - This fit isn’t as good particularly with its patterned residuals, but the atmospheric pressure value is reasonable.
Figure 19: Air Pressure over Time (from Lab Report 2) - This fit isn’t as good particularly with its patterned residuals, and the atmospheric pressure value is a little low.

Figure 20: Air Pressure over Time (from Lab Report 2) - This fit is great, as is the atmospheric pressure value.
6.3 Helium Pressure vs. Time

Figure 21: Helium Pressure over Time - The data fits the model very well. As before, each data point is a single measurement without uncertainty, so the propagated uncertainty and $\chi^2$ values are meaningless.

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>910.75</td>
<td>± 1.38</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.0010594</td>
<td>± 1.55e-06</td>
</tr>
<tr>
<td>$C_2$</td>
<td>-1.1118</td>
<td>± 0.009</td>
</tr>
</tbody>
</table>

$\chi^2 = 142 \ (0.975 \ / \ DoF)$

$P(>) = 0.568$
Figure 22: Helium Pressure over Time - The patterned residuals are very obvious here, but the fit is still good.

Figure 23: Helium Pressure over Time - The patterned residuals are very obvious here, but the fit is still good.
Figure 24: Helium Pressure over Time - This fit is nearly perfect. It is the only graph that doesn’t have the patterned residual. This may be because it was the shortest data run by a factor of two.

Figure 25: Helium Pressure over Time - The patterned residuals are very obvious here, but the fit is still alright.
Figure 26: Helium Pressure over Time 3:10 pm - The patterned residuals are very obvious here, but the fit is still alright.
6.4 Helium Fringes vs. Pressure

Figure 27: Day 2 2:11 PM. Each data point is a single measurement without uncertainty, so the propagated error for the fitting parameters and $\chi^2$ values are meaningless.

Figure 28: Day 2 2:19 PM
Figure 29: Day 2 2:28 PM

Figure 30: Day 2 2:42 PM
Figure 31: Day 2 2:55 PM

Figure 32: Day 2 3:06 PM
6.5 Air Fringes vs. Pressure

Figure 33: Experiment 2 Air Data - 11:49 AM. The data for these plots starts out fitting the curve, but small changes in temperature cause the fringe period to drift and the fit to fail. Over the last 200 mmHg, the fringe period increases substantially. Since each data point is a single measurement without uncertainty, so the uncertainty in the fitting parameters and the $\chi^2$ values are meaningless.

Figure 34: Experiment 2 Air Data - 12:13 PM. This fit starts out well, but the fringe period changes at around 600 mmHg. It appears to match the fit again by 700 mmHg.
Figure 35: Experiment 2 Air Data - 12:52 PM

Figure 36: Experiment 2 Air Data - 1:31 PM

Figure 37: Experiment 2 Air Data - 1:39 PM
6.6 Fourier Transform of Air

Figure 38: Experiment 2 Air Data - 11:49 AM - Spectrum around the peak frequency. The green vertical line is the n-1 calculated by the maximum index method; the red vertical line, if it’s not behind the green line, is the index of refraction calculated by the 3-point average method. Here, the peak is symmetric, so the two methods are nearly equivalent.
Figure 39: Experiment 2 Air Data - 12:13 PM - The red line is closer to the center of the spectra than is the green line.
When the peaks are of nearly equal value, the maximum index method (green) is very inaccurate; the average method (red) is closer to the center.
Figure 41: Experiment 2 Air Data - 1:31 PM - The average method (red) looks like a better estimation of the data’s central frequency than the maximum index method (green).
Figure 42: Experiment 2 Air Data - 1:39 PM - Here, the peak is symmetric, so the two methods are nearly equivalent.
6.7 Fourier Transform of Helium

Figure 43: Day 2 Helium Data - 2:11 PM
Figure 44: Day 2 Helium Data - 2:19 PM

Figure 45: Day 2 Helium Data - 2:28 PM
Figure 46: Day 2 Helium Data - 2:42 PM

Figure 47: Day 2 Helium Data - 2:55 PM
Figure 48: Day 2 Helium Data - 3:06 PM
7 Code Appendix

7.1 process.py

```python
import os
import sys
import numpy as np
import pylab
import math
import smooth
crop = 2

def process(data_fstr, press_fstr, debug_dstr,
fix_press_errors=True, smooth_presses=True):
    print "Processing %s" % os.path.split(data_fstr)[1].split('\\')[0]
    if not os.path.exists(debug_dstr):
        os.mkdir(debug_dstr)

    # use Numpy to load the data from the text file
    data, presses = map(np.genfromtxt, [data_fstr, press_fstr])

    # group the key press locations
    presses = average(group(presses))

    # see if any key presses are missing
    if fix_press_errors:
        presses = fix_presses(presses)

    for p in presses:
        print p

    return

    print len(presses)

    # optionally smooth key press data- USE WITH CAUTION
    if smooth_presses:
        presses = buff(presses, crop)

    savePresses(presses[crop:-crop], os.path.join(debug_dstr, 'presses_raw'), 'Before Smoothing Presses')
    spresses = smooth(smooth(pressess, np.array(presses), window_len=4))
    savePresses(spresses[crop:-crop], os.path.join(debug_dstr, 'presses_smooth'), 'After Smoothing Presses')
    savePresses((spresses - np.array(presses))[crop:-crop], os.path.join(debug_dstr, 'presses_dif'), 'After Smoothing Presses')

    presses = presses[crop:-crop]
    print len(presses)

    saveArray(data, os.path.join(debug_dstr, 'raw'), 'Before Smoothing')

    # Calculate the window length with this estimation. Longer data typically has to be
    # smoothed more! Supersmoothing is used to determine the extrema location. Less
    # smoothing is necessary for the rest.
    windowLen = min(800, len(data) / 80)
    supersmoothdata = list(smooth.smooth(data, window_len=windowLen))
    saveArray(supersmoothdata, os.path.join(debug_dstr, 'supersmooth'), 'After Super Smoothing')

    # Normal smooth the data
    data = list(smooth.smooth(data, window_len=35))
    saveArray(data, os.path.join(debug_dstr, 'smooth'), 'After Smoothing')
```


# Access the voltage where every key was pressed
pressHeights = lookup_vals(data, presses)

# Calculate where the extrema are
minmax, firstIsMax = find_extrema(supersmoothdata, int(presses[0]), int(presses[-1]))
extremaHeights = lookup_vals(data, minmax)
extremas = extrema(minmax, extremaHeights, firstIsMax)
plot_extrema_presses(data, minmax, presses, pressHeights, os.path.join(debug_dstr, 'extrema'))

# Lookup the fractional position of each press between 2 extrema.
# f1 does this in X, and f2 does this in Y.
f1 = crude_fraction(presses, extremas)
f2 = fraction_between_extrema(presses, pressHeights, extremas)
diff = sum(abs(np.array(f1) - np.array(f2))) / len(f2)
print "Average diff between crude & accurate fractions: %.4f wavelengths" % diff
delta2 = find_deltas(presses, f2, extremas)
delta1 = find_deltas(presses, f1, extremas)
mean2, stdev2 = meanstdev(delta2)
mean1, stdev1 = meanstdev(delta1)
delta2 = [0] + list(np.cumsum(np.array(delta2)))
delta1 = [0] + list(np.cumsum(np.array(delta1)))
for i, x in enumerate(delta2):
    print i, x, stdev2, delta1[i], stdev1

""" Calculate mean and standard deviation of data x[]:
mean = \( \frac{\sum_i x_i}{n} \)
std = \( \sqrt{\frac{\sum_i (x_i - \text{mean})^2}{n-1}} \)
"""
def meanstdev(x):
    from math import sqrt
    n, mean, std = len(x), 0, 0
    for a in x:
        mean = mean + a
        mean = mean / float(n)
    for a in x:
        std = std + (a - mean)**2
    std = sqrt(std / float(n-1))
    return mean, std

def find_deltas(presses, frac_extremas, extremas):
    ret = []
    for i in xrange(len(presses) - 1):
        initial = 0.5 - frac_extremas[1]
        final = frac_extremas[i+1]
        extremasIndexAfterInitial = extremas.indexBelowAbove(presses[i+1])[1]
        extremasIndexBeforeFinal = extremas.indexBelowAbove(presses[i+1])[0]
        d = initial + final + (-extremasIndexAfterInitial + extremasIndexBeforeFinal) / 2.0
        ret.append(d)
    return ret
def crude_fraction(presses, extrema):
    ret = []
    for X in presses:
        lowerExtremaIndex, upperExtremaIndex = extrema.indexBelowAbove(X)
        Nbefore = extrema.x[lowerExtremaIndex]
        Nafter = extrema.x[upperExtremaIndex]
        fraction = (X - Nbefore) / (2.0 * (Nafter - Nbefore))
        ret.append(fraction)
    return ret

def fraction_between_extrema(presses, pressHeights, extrema):
    ret = []
    for X, Y in zip(presses, pressHeights):
        beforeExtremaIndex, afterExtremaIndex = extrema.indexBelowAbove(X)
        beforeExtremaYValue = extrema.y[beforeExtremaIndex]
        beforeExtremaMax = extrema.isMax(beforeExtremaIndex)
        afterExtremaYValue = extrema.y[afterExtremaIndex]
        if beforeExtremaMax:
            MIN = beforeExtremaYValue
            MAX = afterExtremaYValue
        else:
            MIN = afterExtremaYValue
            MAX = beforeExtremaYValue
        X = 0.25 + (3 / (2 * math.pi)) * \math.asin((2*Y - MIN - MAX) / (2 * (MAX - MIN)))
        if beforeExtremaMax:
            ret.append(X)
        else:
            ret.append(0.5 - X)
    return ret

def find_extrema(data, start, end):
    first, firstIsMax = find_extrema_left(data, start)
    ret = [first]
extIsMin = firstIsMax
    while first < end:
        first = find_next_extrema(data, first, nextIsMin)
        ret.append(first)
        nextIsMin = not nextIsMin
    return ret, firstIsMax

def find_next_extrema(data, start, isMin, incr = 65):
    s_0 = start + 2 * incr
    if s_0 > len(data):
        print "find_next_extrema error: position %i" % start
s_0 = len(data) - 1
raw_input()
point = data[start]
right = data[start + incr]
right2= data[s_0]
if not isMin and point < right and right > right2:
    return start + index(max, data[start : s_0])
if isMin and point > right and right < right2:
    return start + index(min, data[start : s_0])
return find_next_extrema(data, start+incr, isMin, incr)
def find_extrema_left(data, start, incr = 40):
s_0 = start - 2 * incr
point = data[start]
left = data[start - incr]
left2= data[s_0]
if point > left and left < left2:
    return s_0 + index(min, data[s_0 : start]), False
if point < left and left > left2:
    return s_0 + index(max, data[s_0 : start]), True
else:
    return find_extrema_left(data, start-incr, incr)
def index(function, L):
m = function(L)
for i, val in enumerate(L):
    if m == val: return i
def lookup_vals(data, points):
    return [(1 - p + math.floor(p)) * data[int(math.floor(p))] +
            ( p - math.floor(p)) * data[int(math.ceil(p))] for p in points]
def average(x):
    ret = np.empty(len(x))
    for i in xrange(len(x)):
        ret[i] = sum(x[i]) / 2.0
    return ret
def group(x):
    ret = []
    L = len(x)
    start = None
    for i in xrange(L):
        thisEl = x[i]
        nextEl = x[i + 1] if i+1 < L else None
        if start is None:
            start = thisEl
        if thisEl + 1 != nextEl:
            end = thisEl
            ret.append((start, end))
            start = None
    return ret
```python
def rename(d, n):
    return os.path.join(d, n.replace('.data.txt', '.press.txt'))

def savePresses(n, fname, title=''):  
    xdim = max(len(n) / 600, 8)  
    pylab.figure(figsize = (xdim,6), frameon=False)  
    pylab.plot(n, '.')  
    pylab.ylabel('Press Position')  
    pylab.xlabel('Press Number')  
    pylab.title(title)  
    pylab.savefig(fname + '.png')  
    pylab.close()

def saveArray(n, fname, title=''):  
    raw_plot(n)  
    pylab.title(title)  
    pylab.savefig(fname + '.png')  
    pylab.close()

def plot_extrema_presses(data, minmax, presses, pheights, fname):  
    raw_plot(data)  
    for m in minmax:  
        pylab.axvline(x=m, color='r')  
    pylab.plot(presses, pheights, 'go')  
    pylab.title('Extrema Locations')  
    pylab.savefig(fname + '.png')  
    pylab.close()

def fix_presses(presses, margin=1.7):
    """
    >>> fix_presses([1,2,3,4,5,6,7])
    [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
    >>> fix_presses([1,2,3,5,6,7])
    Skipped press detected at Point 3
    Presses are:
    1 2
    2 3
    INSERTING 4.000000
    3 5
    >>> fix_presses([1,2,3,5,6,7, 9])
    Skipped press detected at Point 6
    Presses are:
    1 2
    2 3
    INSERTING 4.000000
    3 5
    Skipped press detected at Point 6
    """
```
Presses are:
4 6
5 7

INSERTING 8.000000
6 9

[1, 2, 3, 4, 5, 6, 7, 8, 9]

new_presses = [presses[0]]
lastgap = presses[1] - presses[0]

for i in xrange(1, len(presses)):
    newgap = presses[i] - presses[i-1]
    if newgap > lastgap * margin:
        newval = (presses[i] + presses[i-1])/2
        print "Skipped press detected at Point %i" % i
        print "Presses are:"
        print i-2, presses[i-2]
        print i-1, presses[i-1]
        print "INSERTING %f" % newval
        print i, presses[i]
        print "INSERTING %f" % newval
        print i, presses[i]
        new_presses.append(newval)
        newgap /= 2
        new_presses.append(presses[i])
        lastgap = newgap

return new_presses

def buff(L, n=5):

    prv = L[1] - L[0]
nxt = L[-1] - L[-2]
    prv = [L[0] - a * prv for a in reversed(xrange(1, n+1))]
nxt = [L[-1] + a * nxt for a in xrange(1, n+1)]
    return prv + L + nxt

class extrema(object):

def __init__(self, x_points, y_points, firstIsMax):
    self.x = x_points
    self.y = y_points
    self.firstIsMax = firstIsMax

def __len__(self):
    return len(self.x)

def isMax(self, key):
    return key % 2 != self.firstIsMax

def indexBelowAbove(self, x):
    prevvalue = self.x[0]
    for index, value in enumerate(self.x[1:]):
if prevvalue <= x <= value:
    return index, index+1
    prevvalue = value

def main(args):
    join = os.path.join; split = os.path.split
    this_file = args[0]
    parent_dir = split(this_file)[0]
    debug_dir = join(parent_dir, 'debug')
    to_process = [x for x in args[1:] if '.data.txt' in x]

    if to_process == []:
        data = join(parent_dir, 'data')
        files = [join(data, fn) for fn in os.listdir(data)]
        main([this_file] + files)
    else:
        results = [process(f,
                join(rename(*split(f))),
                join(debug_dir, split(f)[1].split('.')[0])
            ) for f in to_process]
        print results

if __name__ == '__main__':
    main(sys.argv)
    raw_input()
7.2 smooth.py

```python
import numpy

def smooth(x, window_len=11, window='hanning'):
    """smooth the data using a window with requested size.
    This method is based on the convolution of a scaled window with the signal.
    The signal is prepared by introducing reflected copies of the signal
    (with the window size) in both ends so that transient parts are minimized
    in the beginning and end part of the output signal.
    input:
    x: the input signal
    window_len: the dimension of the smoothing window; should be an odd integer
    window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'
    flat window will produce a moving average smoothing.
    output:
    the smoothed signal
    example:
    t=linspace(-2,2,0.1)
x=sin(t)+randn(len(t))*0.1
y=smooth(x)
    see also:
    numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve
    scipy.signal.lfilter
    TODO: the window parameter could be the window itself if an array instead of a string
    """
    if x.ndim != 1:
        raise ValueError, "smooth only accepts 1 dimension arrays."
    if x.size < window_len:
        raise ValueError, "Input vector needs to be bigger than window size."
    if window_len<3:
        return x
    if not window in ['flat', 'hanning', 'hamming', 'bartlett', 'blackman']:
        raise ValueError, "Window is on of 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'"
    s=numpy.r_[2*x[0]-x[window_len-1::-1],x,2*x[-1]-x[-1:-window_len:-1]]
    #print(len(s))
    if window == 'flat':  #moving average
        w=numpy.ones(window_len,'d')
    else:
        w=eval('numpy.'+window+'(window_len)')
y=numpy.convolve(w/w.sum(),s,mode='same')
    return y[window_len:-window_len+1]
```

57
from numpy import *
from pylab import *

def smooth_demo():
    t=linspace(-4,4,100)
x=sin(t)
xn=x+randn(len(t))*0.1
y=smooth(x)
ws=31
windows=['flat', 'hanning', 'hamming', 'bartlett', 'blackman']
hold(True)
for w in windows[1:]:
    eval('plot('+'w'+')(ws ))
axis([0,30,0,1.1])
legend(windows)
title("The smoothing windows")

subplot(212)
plot(x)
plot(xn)
for w in windows:
    plot(smooth(xn,10,w))
l=['original signal', 'signal with noise']
l.extend(windows)
legend(l)
title("Smoothing a noisy signal")
show()

if __name__=='__main__':
    smooth_demo()
7.3 pressure resample.py

```python
import os
import sys
import numpy as np
import process
import math

from scipy.interpolate import interp1d

def resample(data_fstr, press_fstr, fit_fstr, dest_fstr, debug_dstr):
    print "Resampling %s" % os.path.split(data_fstr)[1].split('.')[0]
    if not os.path.exists(debug_dstr):
        os.mkdir(debug_dstr)
    if not os.path.exists(fit_fstr):
        print "Save fit parameters for P0 * Sin((t * C1 + C2) / Sqrt(P0))"
        print "Save in %s, in same order as above, one per line" % fit_fstr

    # use Numpy to load the data from the text file
    data, fit, presses = map(np.genfromtxt, [data_fstr, fit_fstr, press_fstr])
    presses = process.average(process.group(presses))
    xstart = presses[0]
    xend = presses[-1]

    @np.vectorize
    def P(x):
        return fit[0] * math.sin((x * fit[1] + fit[2]) / math.sqrt(fit[0]))

    @np.vectorize
    def X(p):
        return (math.sqrt(fit[0]) * math.asin(p / fit[0]) - fit[2]) / fit[1]

    L = len(data)
    interpdata = interp1d(range(L), data)
    pstart = P(xstart)
pend = P(xend)
    pressurePoints = np.linspace(pstart, pend, L)
    resampled = interpdata(X(pressurePoints))
    np.savetxt(dest_fstr, np.column_stack((pressurePoints, resampled)))
    print "Done"

    def rename(f, new):
        [d, n] = os.path.split(f)
        return os.path.join(d, n.replace('.data.txt', '.' + new + '.txt'))

    def main(args):
        join = os.path.join; split = os.path.split
        this_file = args[0]
        parent_dir = split(this_file)[0]
        debug_dir = join(parent_dir, 'debug')
        to_process = [x for x in args[1:] if '.data.txt' in x]

        if to_process == []:
            data = join(parent_dir, 'data')
```

59
files = [join(data, fn) for fn in os.listdir(data)]
main([this_file] + files)
else:
    [resample(f,
        rename(f, 'press'),
        rename(f, 'fit'),
        rename(f, 'resampled'),
        join(debug_dir, split(f)[1].split('.')[0])
    ) for f in to_process]
if __name__ == '_main__':
    main(sys.argv)
7.4 fourier transform.py

```python
import os
import sys
import numpy as np
import process
import math
import process
import pylab
from scipy.interpolate import interp1d
from pressure_resample import rename
from scipy.signal import get_window

def fd(data_fstr, img_fstr, debug_fstr):
    print "Fourier Transforming %s" % os.path.split(data_fstr)[1].split('.')[0]

    print data_fstr
    print img_fstr

    [P, I] = np.loadtxt(data_fstr, unpack=True)[:10]
    Cnmax, nmax, imax, imin, C, Cerr, C2 = fd_worker(P, I, img_fstr)

    print "Cnmax", Cnmax
    print "nmax", nmax
    print "Imax", imax
    print "Imin", imin
    print "C %s +/- %s, %s" % (C, Cerr, C2)

    CtoN = 1609.248193

def fd_worker(P, I, img_fstr):
    N = len(P)
    I -= np.mean(I)
    I *= get_window("blackmanharris", N)
    f = np.fft.fft(I)
    nmax = abs(f[:N/2]).argmax(); Cnmax = abs(f[nmax])
    C0 = abs(f[0])
    imax = 4 * Cnmax / N
    imin = (C0 - 2 * Cnmax)/ N
    C = np.pi * nmax / (P[-1] - P[0])
    Cerr = np.pi / (P[-1] - P[0])
    Cnmaxm = abs(f[nmax-1])**2
    Cnmaxp = abs(f[nmax+1])**2
    Cnmaxs = Cnmax**2
    C2 = ((nmax-1) * Cnmaxm + (nmax) * Cnmaxs + (nmax+1) * Cnmaxp) /
         (Cnmaxm + Cnmaxs + Cnmaxp) * np.pi / (P[-1] - P[0])

    x = C / nmax * CtoN * np.arange(max(0, nmax-5), nmax+5)
    y = abs(f[max(0, nmax-5):max(5)])
    pylab.figure(frameon=False)
    pylab.plot(x, y)
    pylab.plot([C * CtoN] * 2, [0, max(y)*1.1])
    pylab.plot([C2 * CtoN] * 2, [0, max(y)*1.1])
    pylab.ylabel('Amplitude')
    pylab.xlabel('Implied Index of Refraction (n-1)E-6 760mmHg 0C')
    pylab.title("Fourier Transform of Intensity")
    pylab.savefig(img_fstr)
    pylab.close()
```

61
def test_fd_worker():
    C = 0.178
    Phi = 2.038
    Imin = 0.34
    Imax = 1.19
    DeltaP = .015
    PStart = 50
    PEnd = 850;
    P = np.arange(PStart, PEnd, DeltaP)
    I = Imax * (np.cos(2*C*P + Phi) + 1) / 2 + Imin;
    Cnmax, nmax, Imax, Imin, C, Cerr, C2 = fd_worker(P, I, "test.png")
    print("Cnmax", Cnmax)
    print("nmax", nmax)
    print("Imax", Imax)
    print("Imin", Imin)
    print("C %s +/- %s, %s" % (C, Cerr, C2))

def main(args):
    #if 1:
    #    test_fd_worker()
    #    return
    
    join = os.path.join; split = os.path.split
    this_file = args[0]
    parent_dir = split(this_file)[0]
    debug_dir = join(parent_dir, 'debug')
    img_dir = join(join(parent_dir, "img"), "fd")
    to_process = [x for x in args[1:] if '.data.txt' in x]
    
    if to_process == []:
        data = join(parent_dir, 'data')
        files = [join(data, fn) for fn in os.listdir(data)]
        main([this_file] + files)
    else:
        [fd(rename(f, 'resampled'),
            join(img_dir, split(f)[1].split('.')[-1][:-4] + ".png"),
            join(debug_dir, split(f)[-1].split('.')[0])
         ) for f in to_process]

    if __name__ == '__main__':
        main(sys.argv)