Determining Pore Direction and Uniformity of Microchannel Plates

Charles M. Rafkin and Hannah N. Tomio
Advised by Professor Henry Frisch

The Enrico Fermi Institute, University of Chicago

September 2, 2012

Abstract

We investigated how pores in a microchannel plate affect incident light by rotating an MCP and graphing the output of a photodiode receiving light beamed through the plate. We developed a quantitative method to determine the point at which the pores are most aligned with the angle of incident light. After testing at various radii, our method produced data that suggest the pores are aligned evenly throughout the MCP.

Contents

List of Figures

1 Introduction

This paper will detail our investigation of the direction and uniformity of pore alignment in microchannel plates (MCPs). In particular, we will describe our quantitative method of determining both the location of maximum pore alignment and the uniformity of pore alignment. Using a photodiode circuit and a laser pointer, light was shone through the MCP pores and charted the changes in voltage output across the circuit. Since the photodiode produces current proportional to the amount of light shone upon it, the photodiode produces more current (and thus more voltage) when the pores are aligned and less current when the MCP pores block incident light. Using an additional circuit to produce a reference point on the oscilloscope, we were able to calculate the distance from this reference point to the minimum voltage produced on each period.

Some discussion of our basic assumptions is order. To begin with, we have used the absolute sine wave function $y = a\left|\sin\left(\frac{\pi}{L}t + p\right)\right| - k$ to model the data. This is because the pores, if uniformly aligned at a certain radius, will scatter the light as a sine wave. However, rectification and the impossibility of representing "negative light" made it necessary to fit the function to an absolute sine wave function. Although it is a bit nonstandard, we have chosen L and p to represent the period and phase respectively; the function has an angular frequency, then, of $\frac{\pi}{L}$. It should be noted from the outset that not all of our data is a good fit with this model. Ultimately, as our model was very good at predicting the minima of the absolute sine function, the size of χ^2 over the entire curve did not matter to a significant extent. Furthur discussion of the sucesses and failures of our method can be found in Section 7.

The methodology described in this paper proved to be a successful quantitative methodology for determining the pore alignment and uniformity of the MCP. Our method of data collection and analysis of the data allows us to state with some confidence that the pore alignment across the MCP we tested is uniform. Even after taking several tests with different reference points and radii, the data show remarkably little variation in the predicted point of maximum alignment.

2 Objectives

First Objective

To determine direction and uniformity of pores in an MCP quantitatively. This was accomplished using the methodology to be described further in some detail.

Second Objective

To learn basic electronics, programming, and the use of LaTeX. This was a summer project completed by two high schoolers with little prior understanding of electronics or LaTeX. Therefore, we devoted considerable time to preparing external research and attaining necessary grounding in the material, especially Python, with which we had very little facility.

3 Methodology

1. **Preparation and Setup.** The initial setup consisted of placing the MCP on the exact center of a cylinder affixed to a turntable. We moved the laser pointer, fixed at 8° from normal and held by a Lego® arm to point light directly incident upon a photodiode (given no interference from non-aligned MCP pores), to a certain designated radius on the MCP.

2. **Data Collection.** We rotated the turntable across several periods and captured the oscilloscope's readings of the periodic variations in output voltage. Simultaneously, using

the method for finding a reference point detailed in Section 5, the rotating plate established a reference point and the oscilloscope collected the data from periodic pulses corresponding to times in the initial data. We charted and captured both sets of data using the Tektronix Oscilloscope's built in "capture function," which collects 10,000 data points over one image displayed on the screen, no matter the frequency, and saves them as a Comma Separated Value (CSV) file.

3. **Data Analysis.** We ran Python programs main() and refdist(). Using the output from these programs, we generated a four parameter fit (and corresponding chart) for the absolute sine wave fit.[1]

4. **Pore Direction Determination.** Using refdist() to analyze data gathered for many radii across many periods, we calculated the angle of maximum alignment for a variety of data sets. We verified the maximum alignment of the pores for a given radius and determined that the pores were aligned across many radii.

4 Setup

Our setup consisted of a record player, a laser pointer, and Legos®. The MCP rested on a cardboard cylinder with a radius of 1.25 inches mounted upon the record player and secured with duct tape. The top of the cardboard cylinder was covered in electrical tape to create more friction in order to prevent the MCP from shifting during its rotation. To the left of the record player, our primary circuit measured the intensity of the light passed through the MCP. This primary circuit was housed on a Lego®-made "arm" which housed the diode connected to a breadboard that rested underneath the MCP. This circuit was powered by a power supply that sat atop a shelf above the record player and the rest of the setup. Both circuits were connected to an oscilloscope via probes. Our source of light, the laser pointer, was affixed to the Lego® machine at an invariable distance from the laser pointer. Without MCP interference, the laser pointer beamed light directly upon the photodiode.

[1] For brevity, at times the fit will be described as a "sine wave fit." This is because the *method* of fitting is the same for a sine wave and its cousin, the absolute value of the sine wave function. Note, however, that the data do *not* fit very nicely to a sine wave function itself.

Figure 1: Wide View of Setup

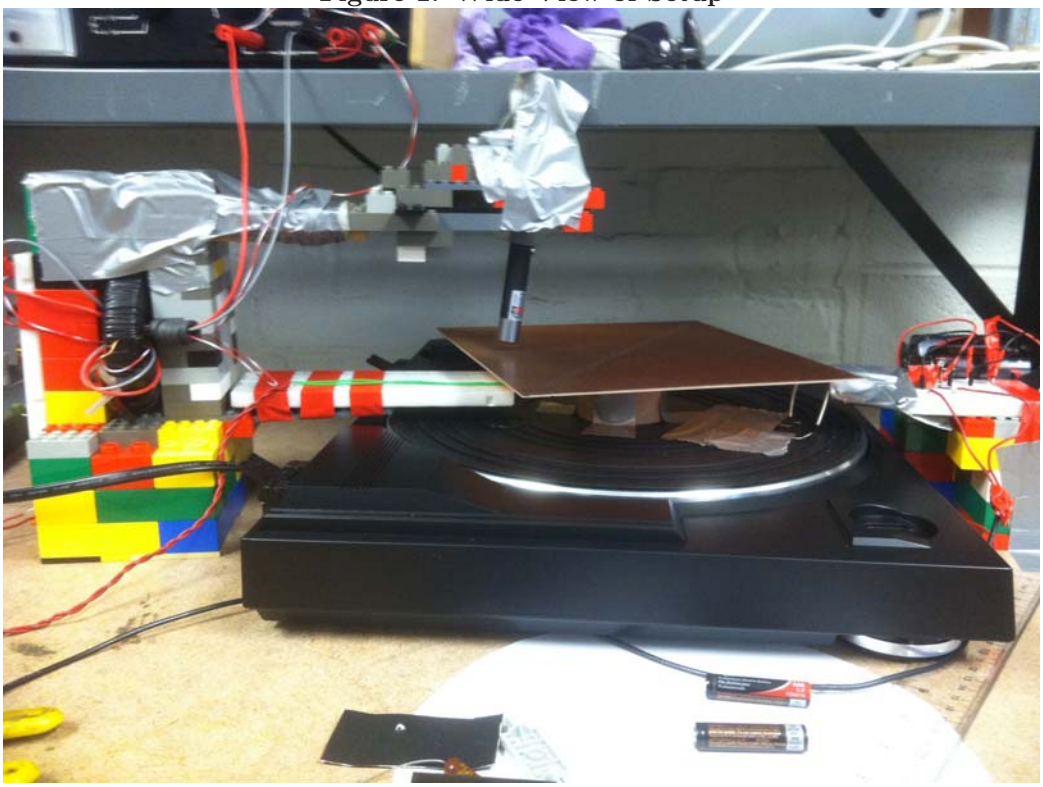
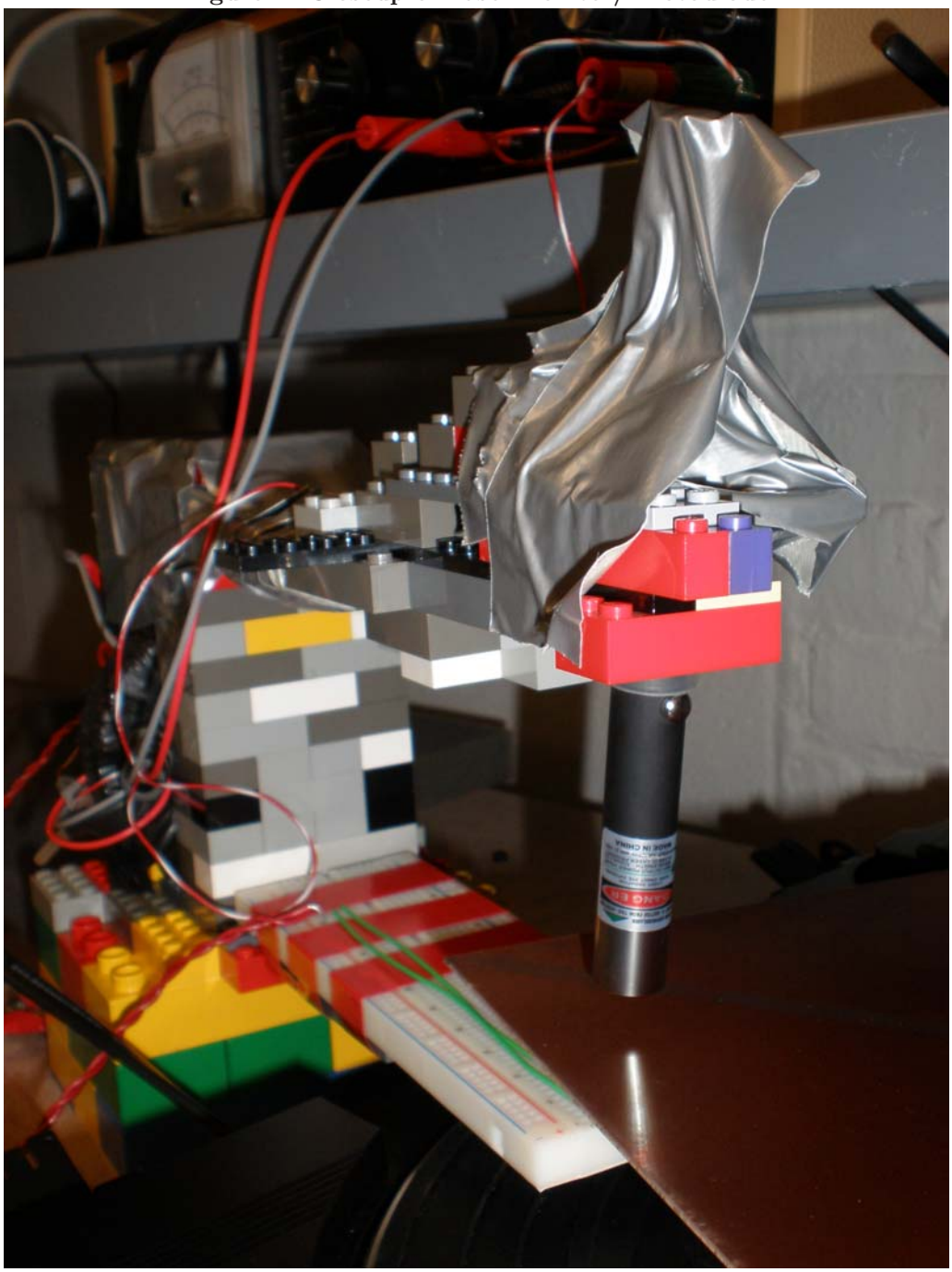


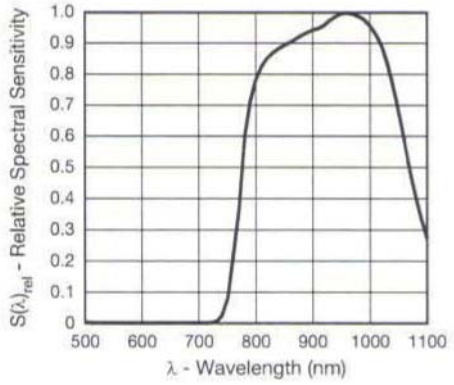
Figure 2: Closeup of Laser Pointer/Photodiode



Part of our setup was adjusted and then returned to its initial state. The record player has only two speed settings; after tinkering with the gears and drive belt, we determined that the manufacturer-set frequencies were, in fact, the record player's optimum frequencies for our purposes. These speeds, $\frac{33\ revolutions}{minute}$ and $\frac{45\ revolutions}{minute}$, were sufficiently slow to allow accurate data reading and prevent MCP slippage on the cylindrical roll but quick enough to produce useful data.

One challenge in the setup involved our attempt to maximize both the photodiode output and the laser wavelength. Our photodiode was most sensitive to light at wavelength $\lambda =$ 950 nm; our laser pointer, however, only produces a wavelength $\lambda = 510$ nm. Due to our circuit design, we were still able to produce sufficient voltage, without "maxing out" the voltage (with respect to the photodiode).

Figure 3: Spectral Sensitivity of Photodiode



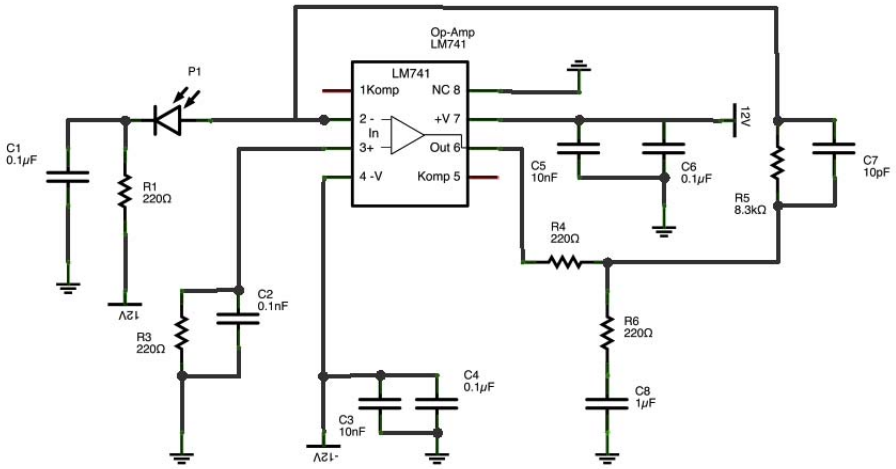
[2]

More significantly, in order to determine the alignment of the pores through data analysis, we needed to establish a reference point for the MCP data. Initially, we placed a thin line of tape directly on top of the MCP. Although this method certainly established a reference point, it caused problems with our curve-fitting program, as the signal spiked toward 0 volts each time the light passed over the black tape and thus interfered with goodness of fit tests. We ultimately removed the tape and created a simple circuit attached to two 1.5 V batteries that only produced a signal at one point during a rotation of the turntable. Further discussion of the circuit processes can be found in the following section.

5 Circuitry

5.1 The Photodiode Circuit

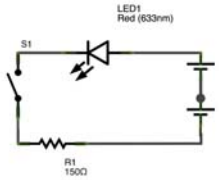
Figure 4: Photodiode Circuit Schematics



The photodiode circuit is used to graph changes in the angle and intensity of incident light as changes in voltage. Our photodiode circuit accepts +12 volts and −12 volts as inputs. The photodiode generates voltage proportional to the amount of light it receives. We included a low-pass filter, R6 and C8, to filter noise. The probe reads the voltage output after C8 and the scope displays the reading on its screen. Due to the circuit design, the output is displayed on the oscilloscope as a negative voltage reading that decreases in value (becomes "more negative") given more incident light.

5.2 The Reference Point Circuit

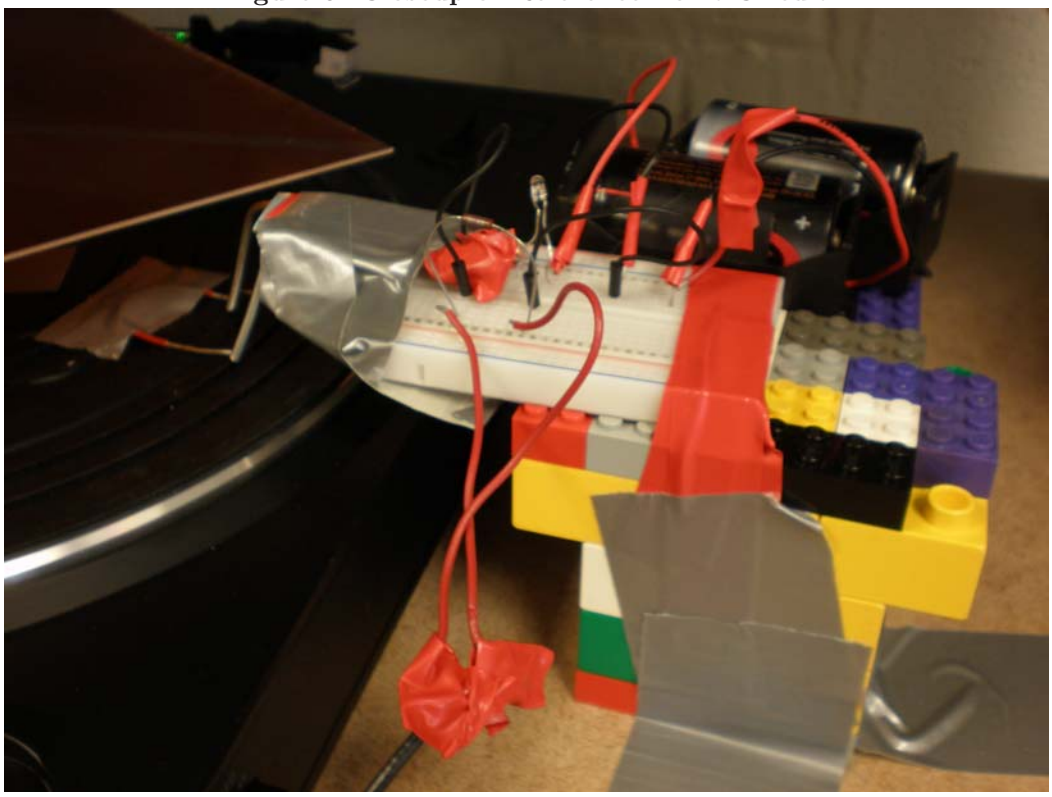
Figure 5: Reference Point Circuit Schematics



Our simple reference point circuit only generates a signal when the "switch" is closed. The "switch" is not a typical on/off switch, but rather a bit of wire attached to the top of the record player, rotating with the same period as the MCP, which only connects the circuit, powered by two "D" batteries, at one point per period. Thus, the signal produced by the circuit registers as a series of pulses, each representing the instant when the turntable rotates past the precise point which closed the circuit. Once we ran our Python programs and determined the distance from the point at which the wire connects the reference point circuit to the minima of the voltage generated, we could determine the angle of alignment of the pores. We know of no circuit symbol

that represents this method of creating a reference point, so we used a switch in the schematic instead.

Figure 6: Closeup of Reference Point Circuit



6 Computer Programming

6.1 Overview

Our program makes use of two outside modules: PyMinuit[3] and PyLab.[4] PyMinuit is a Python module that passes low-level Minuit functionality to Python functions. It proved essential in creating a four parameter fit to our data. PyLab allowed us to perform complicated mathematical operations upon hundreds of (or, alternatively, 10,000) data points and graph the results with ease.

To accomplish our objectives, we used three separate functions: a program that averages groups of data points, a program that fits an absolute sine curve to the data points, and a program that determines the distance from the reference point to the absolute minimum of the data. In order to simplify the interface, the programs were all managed by the main() function. However, since each has an entirely different purpose, they will be discussed separately.

It should be noted that, at this point in our research, the program does *not* compile perfectly. There are significant bugs in the user interface and for around 30% of the tests, the PyMinuit

[3]PyMinuit can be downloaded at https://code.google/p/pyminuit/.
[4]PyLab can be downloaded at https://www.scipy.org/PyLab.

module returns inexplicable errors. Further research ought to be done to correct this malfunction.

6.2 The Central Program

6.2.1 Description

main(), the central program, asks the user a series of questions in order to determine which of the three programs should be run. It is very flexible: the programs can be run in many different combinations depending on the objectives of the user.

We chose to combine all the functions into a single program in order to maximize ease of use. It is simpler for the user to call one function once than to call many functions multiple times to accomplish the same objective. However, there are disadvantages to this approach; if the user only wishes to call one of the functions in order to complete a simple objective, the user will still have to progress through some of main()'s "if" logic.

6.2.2 Code

```
import csv
import math
import tkFileDialog
import Tkinter
import unicodedata

import pylab
import minuit

from collections import defaultdict
from operator import itemgetter

tList = []
yList = []
yEst = []
createtimelist = []
a, L, p, k, c2 = 0, 0, 0, 0, 0

def main():
    Tkinter.Tk().withdraw()
    global tList
    global yList
    apyesorno = ''no''
    avgyesorno = raw_input(''Do you want to average a file?  '').lower()
    fityesorno = raw_input(''Do you want to fit to a curve?  '').lower()
    refyesorno = raw_input(''Do you want to calculate the reference distance?  '
        ').lower()
    if avgyesorno == ''no'' and fityesorno == ''no'' and refyesorno == ''no'':
        graphyesorno = raw_input(''Do you want to print a graph of the data?
            '').lower()
        if graphyesorno == ''yes'':
            thingtograph = tkFileDialog.askopenfilename()
            grapherx = createTList(thingtograph)
            graphery = createYList(thingtograph)
            plt.figure()
            plt.plot(grapherx, graphery, 'c')
            show()
            return
        else:
            print ''Goodbye!''
            return
```

Here, main() determines which of the three functions the user wishes to run by using the built in Python function "raw_input." If the user simply wishes to print a graph of the data, the user may do so by rejecting main()'s requests until the final raw_input.

```
    if avgyesorno == "yes" and fityesorno == "yes" and refyesorno == "yes":
        apyesorno = raw_input("Do you want to average the pulses?   ")
        if apyesorno == "yes":
            print "Please choose the file (of pulses) you'd like to average!"
            pulsestoaverage = tkFileDialog.askopenfilename()
            pulses = Averagecsv(pulsestoaverage)
            print "Next we need the signal file."
    if avgyesorno == "yes":
        print "Please choose the file you want to average."
        toaverage = tkFileDialog.askopenfilename()
        tobefittedfile = Averagecsv(toaverage)

        tList = createTList(tobefittedfile)
        yList = createYList(tobefittedfile)
        tListancient = createTList(toaverage)
        yListancient = createYList(toaverage)
        plt.figure()
        plt.plot(tListancient, yListancient, 'c')
        plt.plot(tList, yList, 'k')
        show()
        if fityesorno == "no":
            return tobefittedfile
    if avgyesorno == "no":
        print "Please choose the file you want to fit to sine."
        tobefittedfile = tkFileDialog.askopenfilename()
        tList = createTList(tobefittedfile)
        yList = createYList(tobefittedfile)
    filereader = csv.reader(open(tobefittedfile, 'r'), delimiter = ',')
    fitter(filereader)
    if refyesorno == "yes":
        if apyesorno == "yes":
            refdist(pulses)
        else:
            refdist(None)
```

After determining which of the programs it should to run, main() asks the user to select the files in question (using Python module Tkinter) and runs the programs.

6.3 The Averaging Program

6.3.1 Description

The first function we designed, called AverageCSV(), computes the average of a group of user-defined datapoints. Since the Tektronix Oscilloscope produces a 2-column dataset with the first column representing the time and the other indicating the corresponding voltage, we designed the program to accept a 2-column CSV. The program asks the user to select the document that needs to be averaged and saves a new CSV file that averages both the time and voltage values in a user-specified range. For instance, given a set of 10,000 data points, if the user commands AverageCSV() to average 50 data points, the program will produce a set of 200 data points. The first datapoint (t, V)in the new CSV file represents (the average of the first 50 t values, the

average of the first 50 V values). If the total number of initial data points is not evenly divisible by the number chosen as the amount of data points to average (i.e., if one chose to average every 35 points of a total of 10,000), the program averages the remainder and prints it as the final data point in the set.

The program proved useful in noise-reduction and helped create a better fit with main(). Although we did reduce a significant amount of the noise by modifying our setup (using toroids, shorter wires, better circuit design, etc.), noise reduction proved difficult due to the noise produced by the motor of the record player. Newer oscilloscope models have similar programs built into their interface for this purpose.

6.3.2 Code

```
1
2  def AverageCSV(toaverage):
3      Tkinter.Tk().withdraw()
4      while True:
5          try:
6              avgnum = int(raw_input(``How many data points do you want to average
                   together?       ''))
7              break
8          except ValueError:
9              print ``Please enter a number!''
10     finalfile = raw_input(``What should the name of the new file be?     '')
11     print ``Where should this file be saved?     ''
12     saveplace = tkFileDialog.askdirectory()
13     csvfile = csv.reader(open(toaverage, 'rb'), delimiter=',')
14     newfile = open(saveplace+'/'+finalfile, 'w')
15     csvwriter = csv.writer(newfile, delimiter = ',')
16     count = []
17     for row in csvfile: #changes your datapoints to a single list,with collumn 1
           being count evaluated at even numbers and collumn 2 being the count
           evaluated at odd numbers
18         count += row
19     for n in range(0, len(count), 2*avgnum): #this creates data points again, and
           averages both collumns (in groups of however many you input)
20         sum = [0.0, 0.0]
21         if n + (2*avgnum-1) <= len(count):
22             for i in range(0, 2*avgnum, 2):
23                 sum = [(sum[0] + float(count[n + i])), (sum[1] + float(count[n + i
                       +1]))]
24             average1 = [sum[0]*  float(avgnum)**(-1), sum[1] * float(avgnum)**(-1)
                   ]
25             csvwriter.writerow(average1)
26         elif n + (2*avgnum-1) > len(count): #for the last few datapoints, might
               not be exact number correct left over. this takes the average of those
                left left and puts them in correct collumn.
27             remainder = len(count) - n
28             for j in range(0, remainder - 1, 2):
29                sum = [(sum[0] + (float(count[n + j]))), (sum[1] + float(count[n +
                       1 + j]))]
30             average2 = [ ( sum[0]*  float((remainder * .5)**(-1))), (sum[1] *
                   float((remainder * .5)**(-1)))]
31             csvwriter.writerow(average2)
32     return saveplace + ``/'' + finalfile
```

6.4 Fitting Program

6.4.1 Description

fitter(), the Sine Fitting Program, accepts a CSV file and minimizes the chi-square of the best four parameter fit of the function $y = a\left|\sin\left(\frac{\pi}{L}t + p\right)\right| - k$ to the list of averaged data. The chi-square goodness of fit test is defined as $\chi^2 = \sum_{k=1}^{n} \frac{(y_{measured} - y_{predicted})^2}{(variance)^2}$. When the chi-square is minimized, the output parameters a, L, p, and k best fit the function to the data.

There are two main sections of fitter(). The first section interprets the data and determines good estimates for the PyMinuit function migrad(), which will minimize the chi-square. The second section runs migrad() and minimizes the four parameter chi-square fit. PyMinuit is not perfect: it can erroneously claim a relative minimum for the chi-square is the best fit. The risk of PyMinuit malfunctioning is reduced significantly if it is given good estimates.

fitter() has some bugs that should be worked out in the future. In particular, the program is designed under the assumption that it is being fed files that have already been returned by AverageCSV(), and so cannot fit a function to the *measured* data, because it gets thrown off quite easily by outliers. It also assumes the data looks like ours (with a negative vertical shift, a period between 1 and 2, and data that *looks* much like an absolute sine wave to begin with).

6.4.2 Code

```
1
2  def createTList(filename): ## given a path to a csvfile, this function returns a
       list of the t values (x values) in the csvfile
3      csvfile = csv.reader(open(filename, 'rb'), delimiter = ',')
4      count = []
5      tlist = []
6      for row in csvfile:
7          count += row
8      for point in range(len(count)):
9          if point % 2 == 0:
10             tlist += [count[point]]
11     return tlist
12
13 def createYList(filename): ## given a path to a csvfile, this function returns a
       list of the y values in the csvfile
14     csvfile = csv.reader(open(filename, 'rb'), delimiter=',')
15     count = []
16     ylist = []
17     for row in csvfile:
18         count += row
19     for point in range(len(count)):
20         if point % 2 == 1:
21             ylist += [count[point]]
22     return ylist
```

These two simple functions accept a CSV file that is usually created after the user runs main() to locate the file to be fitted to a sine wave. createTList() returns a list of the time values. createYList() returns a list of the voltage values. These functions, like the rest of the functions other than AverageCSV() and refdist(), are not intended to be called outside of main().

```
1  def createsortedy(xlist, yListorig, count): ## given a list of x values, a list of
       y values, and a list containing both x and y values, this function sorts the
       list containing both values by the y values
```

```
    counter = []
    bigcount = []
    for row in range(1, len(count), 2):
        bigcount += [(float(count[row - 1]), float(count[row]))]
    newsortedlist = sorted(bigcount, key = itemgetter(1))
    timeusable = float(newsortedlist[0][0])
    yusable = float(newsortedlist[0][1])

    return timeusable, yusable, newsortedlist, float(newsortedlist[0][0])
```

This function uses a method involving Python dictionaries and tuples to sort a list of ordered pairs (x, y) by the y values. For example, given the x list (1, 2, 5) and the y list (3,2,7), the function assumes the pairs are linked and sorts them as [(2,2), (1,3) (5,7)]. It returns the time at the absolute minimum of the data, the y value at the minimum, the list of sorted values, and the time at the absolute minimum of the function (later defined as the variable "timezero"). The time at the absolute minimum is returned twice because any data point could be chosen to represent "timeusable" and "yusable," but the time value at the minimum is necessary, regardless of the point chosen, to calculate an estimate for the phase.

```

def chi2(a, L, p, k): ## creates a chi-square from global tList and yList using a
    given amplitude, period, phase, and vertical shift
    global c2
    c2 = 0
    for i in range(len(tList)):
        c2 += abs(((float(yList[i]) - (a * float(abs(sin(pi * (L**(-1)) * float(
            tList[i])+p))) - float(k)))**2) * (25))
    return c2
```

This function returns the chi-square of the four-parameter fit of the function $y = a\left|\sin\left(\frac{\pi}{L}t + p\right)\right| - k$. Note that since our circuit design produces a negative value for the vertical shift, k is defined in main() as positive in order to produce an accurate vertical translation of the function. The variance estimate is assumed to be random over all points and is estimated at around 200 mV, or .2 V (note: $.2^2 = .04$). Although there are some problems with the chi-square test, especially its assumption that errors will vary normally, it provided a superior fit.

```
def fitter(csvfile):
    count = []
    helper = []
    period = 0
    timelist2 = []
    time2 = 0
    for row in csvfile:
        count += row
    timeusable, yusable, newsortedlist, timezero = createsortedy(tList, yList,
        count)
    for point in range(len(count)):
        if abs(float(count[point - 1]) - timeusable) > 1 and abs(float(count[point
            - 1]) - timeusable) < 2 and point % 2 != 0:
            timelist2 += [(float(count[point - 1]), abs(float(count[point]) -
                yusable))]
    sortedfinalpoints = sorted(timelist2, key = itemgetter(1))
    if len(timelist2) >= 1:
        time2 = float(sortedfinalpoints[0][0])
```

```
    if len(timelist2) == 0:
        while True:
            try:
                rangedistance = float(raw_input(''Give a bigger range distance!
                    ''))
                break
            except ValueError:
                print ''Please enter a number!''
        for point in range(len(count)):
            if abs(float(count[point]) - yusable) < rangedistance and abs(float(
                count[point - 1]) - timeusable) > 1 and abs(float(count[point -
                1]) - timeusable) < 2 and point % 2 != 0:
                timelist2 += [(float(count[point - 1]),abs(float(count[point])) -
                    yusable)]
        sortedfinalpoints = sorted(timelist2, key = itemgetter(1))
        time2 = sortedfinalpoints[0][0]

    period = abs(timeusable - time2)
    phase = -pi * (period**-1) * timezero
```

Here, fitter() accepts the output variables from createsortedy() and uses them to find a distance from local minimum to local minimum in the data. This distance, the period, is set as the variable "period." An estimate for the phase can be found given the time when the function is minimized. Given the function $y = a\left|\sin\left(\frac{\pi}{L}t + p\right)\right| - k$, if t_n equals the minimum of the function, then when $y = -k$:

$$\begin{aligned} y &= a\left|\sin\left(\frac{\pi}{L}t_n + p\right)\right| - k \\ 0 &= a\left|\sin\left(\frac{\pi}{L}t_n + p\right)\right| \\ 0 &= \frac{\pi}{L}t_n + p \\ p &= -\frac{\pi}{L}t_n \end{aligned}$$

Through this method, fitter() computes the phase using the previously determined values for t_n and L.

```
    amax1 = []
    amin1 = []
    period2 = int(period * abs(((float(tList[0])) - float(tList[1]))**(-1)))

    for n in range(0, len(yList) - period2, period2):
        amax2 = float(yList[0])
        amin2 = float(yList[0])
        for j in range(n, n + period2):
            if amax2 < float(yList[j]):
                amax2  = float(yList[j])
            if amin2 > float(yList[j]):
                amin2 = float(yList[j])
        amin1 += [amin2]
        amax1 += [amax2]

    sum1 = []
    sum2 = 0
    for i in range(len(amax1)):
        sum1 += [abs(float(str(amax1[i])) - float(str(amin1[i])))]
```

```
20      for i in range(len(sum1)):
21          sum2 += float(sum1[i])
22      vertshifthelper = 0
23      for i in range(len(amax1)):
24          vertshifthelper += float(str(amax1[i]))
25      myvar = vertshifthelper * len(amax1)**-1
26      amplitude = sum2 * (len(amax1)**(-1))
27      vertshift = (amplitude + abs(myvar))
```

In this section, fitter() computes the average of the amplitudes (omitting the final one because it created such problems with the remainder that we relied on PyMinuit to smooth out the errors), defined as the distance between minima and maxima. Then, as it can be shown that $k \approx -\frac{a}{2}$, [5] main() uses its calculation of the amplitude to determine an estimate for the vertical shift.

```
1       m = minuit.Minuit(chi2)
2       m.strategy = 0
3       m.values[''a''] = amplitude
4       m.values[''L''] = period
5       m.values[''p''] = phase
6       m.values[''k''] = vertshift
7
8       m.printMode = 1
9       m.migrad()
10
11      m.hesse()
12      aerr = m.errors[''a'']
13      Lerr = m.errors[''L'']
14      perr = m.errors[''p'']
15      kerr = m.errors[''k'']
16      global a
17      global L
18      global p
19      global k
20      global yEst
21      global createtimelist
22      a, L, p, k = m.values[''a''], m.values[''L''], m.values[''p''], m.values[''k''
            ]
23      createtimelist = arange(float(tList[0]),float(tList[-1]),.01)
24      yEst = []
25      for t in createtimelist:
26          yEst += [a * abs(sin(pi * (L**(-1)) * float(t) + p)) - abs(k)]
27      plt.figure()
28      plt.plot(tList, yList, 'b')
29      plt.plot(createtimelist, yEst, 'r')
30      title(''y = '' + str(round(a, 3)) + ''|sin((pi/'' + str(round(L, 3)) + '')*x +
            '' + str(round(p, 3)) + '')| - '' + str(round(k, 3)) + '' chisquared = ''
            + str(round(c2, 3)))
31      sign = unicodedata.lookup('PLUS-MINUS SIGN')
32      print ''amplitude = '' + str(round(a, 3)) + sign + str(round(aerr, 3)) + ''
            period = '' + str(round(L, 3)) + sign + str(round(Lerr, 3)) + '' phase = ''
            + str(round(p, 3)) + sign + str(round(perr, 3)) + '' vertical shift = ''
            + str(round(k, 3)) + sign + str(round(kerr, 3)) + '' chisquared = '' +
            str(round(c2, 3))
33      show()
```

Next, fitter() creates a Minuit object and feeds migrad() its estimates for the amplitude, period, phase, and vertical shift. migrad() then determines the minimum chi-square for those parameters

[5] k should be around equal to the y coordinate of the midpoint of between the minima and the maxima.

and hesse() returns an error estimate for each parameter. MatPlotLib graphs the fitted curve alongside the averaged data, and Python prints the estimates for the four parameters, the error estimate, and the chi-square.

6.5 Reference Point Program

6.5.1 Description

The reference point program refdist() determines the average distance for each period from the pulse created by our reference point circuit to the minima of the function, when the MCP pores allow most of the incident light to pass through to the photodiode. There were some challenges in designing the program. It is difficult to find the time values for periodic minima in Pylab and to generate proper values for the list of points evaluated at the fitted function. In addition, because they occur at different points in one period of the function, it is possible to have more of either pulses or minima than the other, which creates problems appending lists when iterating over set ranges. However, we were eventually able to write a program that, although not bug free, was sufficient for our purposes. As this code is also relatively lengthy, more specific comments will again be presented after each major portion of the function.

6.5.2 Code

```
1
2
3 #overlay the averaged reference point pulses onto fitted curve of data and
      determine the average time from the reference point to the minima of the
      fitted function.
4
5 def refdist(pulses):
6     Tkinter.Tk().withdraw()
7     if pulses is None:
8         print ''What is the name of the averaged pulses you want to overlay?
                  ''
9         pulses = tkFileDialog.askopenfilename()
10    xpulses = createTList(pulses)
11    ypulses = createYList(pulses)
12    plt.figure()
13    plt.plot(xpulses, ypulses, 'r')
14    plt.plot(createtimelist, yEst, 'c')
15    show()
```

Initially, refdist() asks the user to select the file of pulses the user wishes to use as a reference point and graphs the reference points alongside the function itself.

```
1     listofpulses = []
2     for x in range(len(xpulses)):
3         listofpulses += [xpulses[x]] + [ypulses[x]]
4     while True:
5         try:
6             lowerbound = float(raw_input(''Give a lower bound for the size of the
                  pulses                  ''))
7             break
8         except ValueError:
9             print ''Please enter a number!''
10
```

```
    bigcount = []

    for a in range(1, len(listofpulses), 2):
        bigcount += [(float(listofpulses[a-1]), float(listofpulses[a]))]
    newsortedlist = sorted(bigcount, key = itemgetter(1))
    unsortedsixxpulses = []
    for i in range(len(newsortedlist)):
        if float(newsortedlist[i][1]) > float(lowerbound):
            unsortedsixxpulses += [newsortedlist[i][0]]
    sixxpulses = sorted(unsortedsixxpulses)
```

refdist() accepts a CSV file of pulses averaged for *the same number of points* as its companion file which measures the output from the photodiode circuit and creates a list of tuples (x,y) of points greater than a certain user-designated value (the size of the pulse can vary). It then creates a new list of time values for the reference point pulses sorted in ascending order.

```
    helper = []
    functionminimumtimes = []
    sinewavelist = []
    for x in arange(float(xpulses[0]),float(xpulses[-1]),.0001):
        sinewavelist +=  [(x, a*abs(sin(pi*(L**(-1))*x + p)) - abs(k))]
    absoluteminimum = float(sorted(sinewavelist, key = itemgetter(1))[0][0])
    mylist1 = []
    mylist2 = []
    for i in arange(absoluteminimum, float(xpulses[-1]), float(L)):
        mylist1 += [i]
    for n in arange(absoluteminimum, float(xpulses[0]), -float(L)):
        mylist2 += [n]
    sortedsixsinewavepoints = sorted(mylist1 + mylist2[1:])
```

For every .0001 seconds, refdist() evaluates the function $y = a\left|\sin\left(\frac{\pi}{L}t + p\right)\right| - k$ within the total domain of the dataset. Then, using the absolute minimum of these evaluated values and the period, the function writes a list of the time values at each of the minima.

```

refdist()
    print ''the times when it pulses are: '' + str(sixxpulses)
    print ''the times when it is at a minimum are '' + str(sortedsixsinewavepoints
        )
    print ''pulses length = '' + str(len(sixxpulses))
    print ''sortedsixsinewavepoints length = '' + str(len(sortedsixsinewavepoints)
        )
    if len(sixxpulses) != len(sortedsixsinewavepoints):
        print ''WATCH OUT! IT'S LIKELY SOMETHING WENT WRONG - THE LENGTHS ARE
            DIFFERENT''
    myrange = min(len(sixxpulses), len(sortedsixsinewavepoints))

    for x in range(myrange):
        helper += [float(sortedsixsinewavepoints[x]) - float(sixxpulses[x])]

    if sum(helper) >= 0:
        average = float(sum(helper)*((myrange)**(-1)))
    if sum(helper) < 0:
        average = float(L - abs(sum(helper)*((myrange)**(-1))))

    standarddeviation = std(helper)
```

```
20      print ``average x distance from pulse to minimum = '' + str(round(average,4))
            + `` or '' + str(round(((average*360)*(L**(-1))), 4)) + `` in degrees''
21      print ``the standard deviation of the differences between pulse and minima is
            '' + str(round(standarddeviation, 6)) + `` or '' + str(round(((
            standarddeviation*360)*(L**(-1))), 4)) + `` in degrees''
```

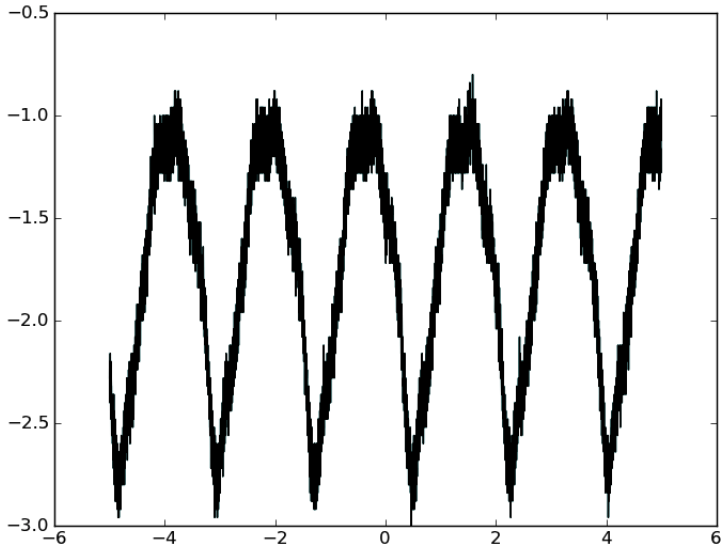
Finally, the function calculates and prints the average difference and standard deviation in time between the pulses and minima of the function and converts the difference into to degrees. Because of a small chance of some bugs arising, the function writes certain error warnings and the lists of the times of the pulses and minima to ensure the user is aware of possible issues should they arise.

7 Determining Pore Direction and Uniformity

7.1 A Closer Look at the Data

After rotating the MCP and capturing a set of periods, the readings on the oscilloscope usually resembled the following:

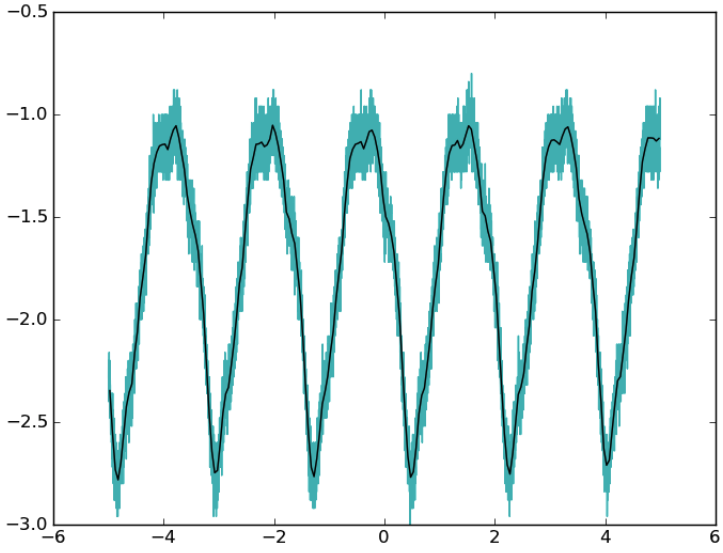
Figure 7: An Example Test



As is evident by Figure 7, the data are noisy, but readable: it is identifiable that the data fit to *some* function at a certain period, although it is unclear if the function fits very well to a sine wave.

After averaging every 50 data points, our data produced the following chart:

Figure 8: An Averaged Example Test

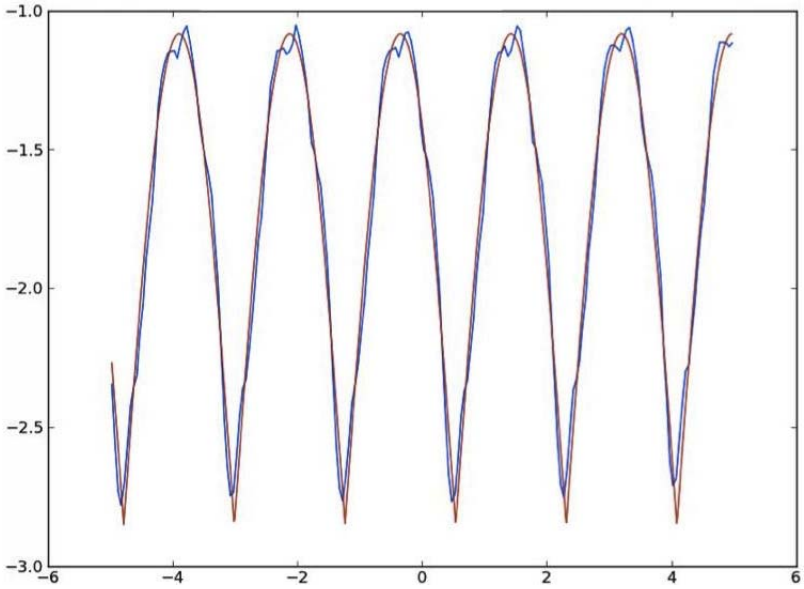


The plot of the data after every fifty points have been averaged together is overlaid in black upon the data plotted in light green. It is clear that the averaged data produce a more easily comprehensible graph. Beyond simple readability and ease of understanding, averaging the data reduces the effects of outliers, a necessity for fitter().

7.2 Modeling the Data

Using our model, we were able to generate signals and chart data averaged using the program AverageCSV() alongside the model generated using fitter(). In some cases, our model produced very accurate results. For example, using a different data set:

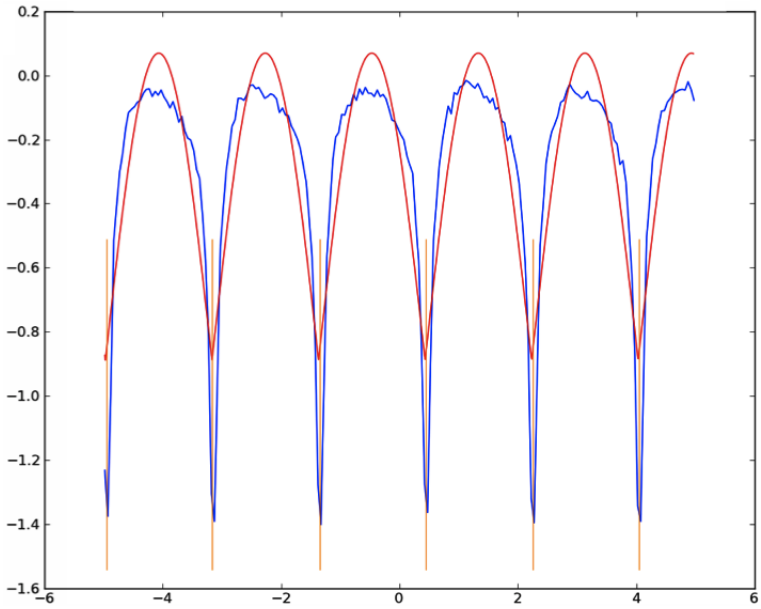
Figure 9: $y = 1.768\left|\sin\left(\frac{\pi}{1.774}t + 8.473\right)\right| - 2.852$ **;** $\chi^2 = 31.601$



As Figure 9 makes clear, the χ^2 for the averaged data is relatively low; in this case, the average error $\approx$ .066 volts. The model is very accurate: for the vast majority of points on the curve, with the notable exception of the minima, the function can accurately predict the output of the circuit.

However, our model was a poorer fit for other data. For example:

Figure 10: $y = .959\left|\sin\left(\frac{\pi}{1.799}t + 2.387\right)\right| - .889$ **;** $\chi^2 = 151.963$



At this point in our research, it is unclear why the model fits so poorly to the data for this test. The test cannot be discarded as an exception; for many tests, especially after we finalized the setup and methodology, the model is not a good predictor of output voltage.

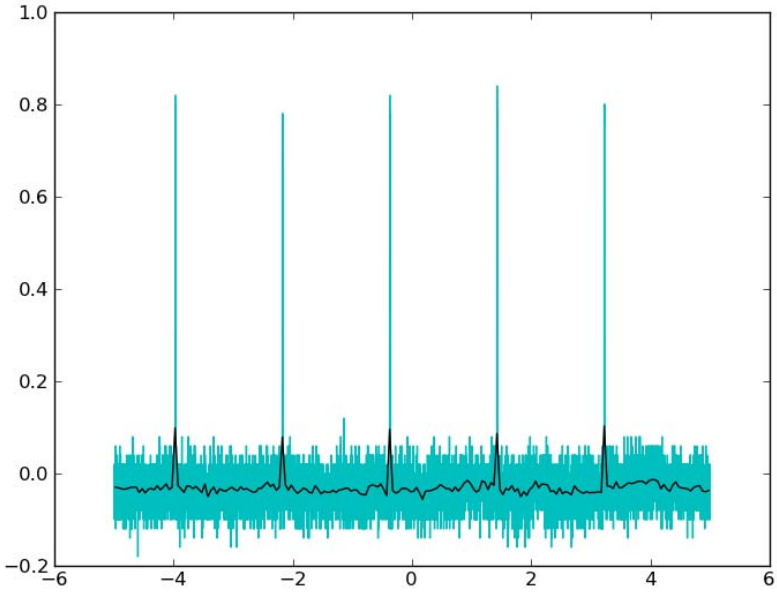
However, this does not mean the model should be discarded. The purpose of this paper, after all, was *not* to find the perfect model of the data; were we to do so, we would likely need to add separate trigonometric terms using Fourier methods to come to a more precise model for the data. Rather, we sought to predict the distribution and uniformity of the pores, for which we solely required a good indicator of the period. As demonstrated by Figure 2, the model does give a good estimate for the period: the yellow vertical lines, added afterward in Photoshop, should illustrate that the minima of the data correspond very closely with the minima of the function.

One improvement suggested to us by our advisor Professor Henry Frisch was that the model may be "upside-down." That is, the function might not be best modeled by the absolute value of the sine wave, but rather by a rectified sine wave (with outputs of negative voltage, rather than positive voltage). Although this concept does seem intuitively logical – the function should not be producing values at all when it is receiving "negative light" – it may not be supported very well by our data, which do seem to fit to some sort of adjusted sine wave. We therefore recommend further investigation into the exact nature of the scattering of the light by the MCP pores.

7.3 Results

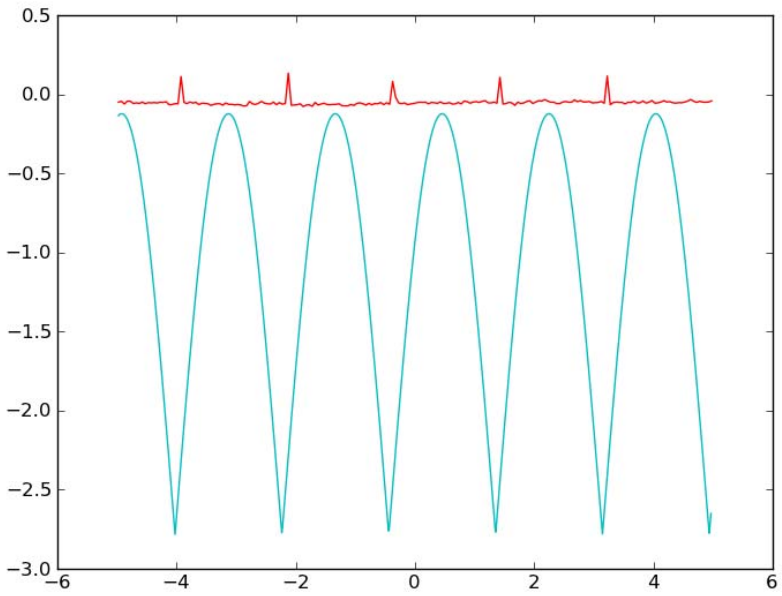
Having developed a model for the data, we sought to overlay the model with the measured reference point pulses. Recall our method for creating a reference point: we built a separate circuit that closed only once every period of the turntable rotation and measured the voltage output of this circuit. The voltage output, then, registers as a series of pulses. Each pulse represents the instant of time during which the circuit was closed and thus when the turntable had rotated past a certain point. The output registered as such:

Figure 11: Reference Point Signal Overlaid Upon Photodiode Signal



Having established a reference point, we overlaid the the battery-generated reference point pulses upon the photodiode circuit's data for the same reading:

Figure 12: Pulse Signal Averaged by AverageCSV()



From the data, the program refdist() will determine the distance from reference point to minima of the fitted absolute sine function. It will print this average distance both as a time, in seconds, and as an angle as measured from the center of the MCP. The printout represents the distance from the reference point to the minima. Taken over many periods, the program will also provide the standard deviation of the distance from reference point to minima.

After the first trial with an arbitrary reference point, our method produced the following results, as analyzed by refdist(). It ought to be noted that for all following results, our confidence in the results produced by refdist() has been compromised due to the aforementioned PyMinuit bug that unfortunately prevented running main() on certain averaged data. PyMinuit would run, however, if the data were averaged for a different number; for example, if PyMinuit would not compile the data averaged for every 50 points, we tried running PyMinuit on the data averaged for every 35 points (or 100 points, etc.). This is not a perfect method of solving the bug: on data PyMinuit analyzed properly, changing the number of points averaged together by AverageCSV() did change the predicted angle of maximum alignment, albeit sightly. In other words, in prioritizing PyMinuit, we have introduced another independant variable that interferes with the data. However, in an effort to present our collected data, we have chosen to display the data regardless. The data averaged by a number other than 50 (our standard averaging number) is marked with a *. Additionally, during the course of Trial 1 although we were able to analyze each of the data, we did not collect data at 2.75 inches.

Figure 13: Trial 1 Data Collection

Trial Number	Radius (inches)	Predicted Angle of Maximum Alignment	Standard Deviation of the Predicted Angle
1	2.25	318.528°	0.9628°
	2.50	301.1093°	0.2674°
	3.00	321.6216°	0.1806°
	3.25	323.0961°	2.68°
	3.50	331.8311°	2.84°
Standard Deviation		11.27°	

Although imperfect, the data in Trial 1, presented in Figure 12 have a standard deviation of only 11.27°, or around 3.1%.[6] Further trials, each with a different reference point, produced even stronger data:[7]

Figure 14: Trials 2-4 Data Collection

Trial Number	Radius (inches)	Predicted Angle of Maximum Alignment	Standard Deviation of the Predicted Angle
2	1.50	340.0°	0.0°
	1.75	342.8489°	3.2046°
	2.00	340.0°	0.0002°
	2.25	339.2867°	2.3735°
	2.50	346.6492°	3.1044°
	2.75	339.9999°	0.0°
Standard Deviation		2.83°	
4	1.25	350.2739°	2.5068°
	1.50*	345.7405°	5.225°
	1.75	348.8171°	2.5711°
	2.00	349.1841°	3.0863°
	2.25*	347.2011°	1.9633°
	2.50	350.9265°	2.8887°
	2.75*	353.4072°	5.4967°
	3.00*	354.1921°	4.457°
	3.25*	355.8248°	2.9145°
	3.50*	355.478°	1.3287°
	3.75	351.5267°	3.2822°
Standard Deviation		3.32°	

[6] Of a full 360°.

[7] During the course of Trial 3, the MCP was knocked and reference point shifted and thus only three data points were gathered. These points were not analyzed using the computer program – regardless of the results, three points would not be enough to establish either a pattern or a reversal of a pattern. Therefore, we have chosen to omit this trial.

The data suggest that the angle of maximum pore alignment does not vary with the radius, but instead remains at a constant angle throughout the MCP. As should be made clear by Figure 14, the standard deviations for the predicted angle of maximum pore alignment for both Trial 2 and Trial 4 are both below 3.5°for a number of radii. The variation in the predicted angle of maximum alignment appears sufficiently low as to be explained by inaccuracies in our methodology and programming analysis, rather than by non-uniformity in pore alignment.

8 Conclusion

We have attempted to illustrate our quantitative method for determining the angle of maximum pore alignment for an MCP and have explained our use of this method to furnish evidence that the pores in an MCP are aligned throughout its radii. Given that the completion of this project was impossible due to a variety of constraints, we offer the following recommendations for further research:

Improve Setup Design and Methodological Accuracy. As Figure 1 and Figure 2 indicate, our setup, partially comprised of Legos® and duct tape, may result in built-in imprecisions. For example, our use of Legos® made perfectly aligning the laser pointer to shine light upon the photodiode exceedingly difficult; it is uncertain that the MCP was centered directly upon the cardboard cylinder resting on the turntable (which was, itself, likely imperfectly centered); the motor from the turntable generated enough noise to require us to build an external computer program to eliminate it (and thus reduce accuracy); our reference point circuit did not merely produce a signal for one data measurement; etc. There were also methodological inaccuracies that could be eliminated, particularly the inaccuracy in measurement of the incident light from the photodiode upon the MCP (as the precise location of the light was difficult to determine).

Fix main(). That main() inexplicably throws PyMinuit errors for certain readings of the oscilloscope is deeply troubling. However, after struggling with the code for many hours, we simply have not been able to come to a perfect solution to the Python errors. We feel confident that those with more extensive computer science background would be able to resolve the errors quickly. Our solution to the problem – simply to input a different value into AverageCSV() – is imperfect at best, as different values inputted into AverageCSV() change the inputs for fitter and thus create slightly different predicted minima, introducing another source of uncertainty into refdist().

Improve fitter(). As has been detailed at length throughout the paper but particularly in section 7.1, our model did not consistently provide a good (or even passable) fit to the data. Although we were able to work around the model's substantial problems, a model with a much-reduced χ^2 – and an explanation for the theoretical grounding of the model – would be a substantial addition to our understanding of the MCP pore distribution.

Collect More Data. Time constraints in particular have prevented us from performing a sufficient number of trials to truly establish the uniformity in alignment of the MCP pores. The three trials we have presented should merely be used as a springboard for further inquiry.

References

[1] Frisch, Henry. "The Development of Large-Area Pico-second Photodetectors." Presentation, SLAC National Accelerator Laboratory, June 15, 2012, accessed July 2 2012, http://hep.uchicago.edu/~frisch/talks/SLAC_v7.pdf.

[2] Horowitz, Paul, and Winfield Hill. *The Art of Electronics.* New York: Cambridge University Press, 1989.

[3] Vishay Semiconductors, *Silicon PIN Photodiode*, Fig. 5 - Relative Spectral Sensitivity vs. Wavelength, November 29, 2011, accessed July 2, 2012, http://www.ml.com/annualmeetingmaterials/2007/ar/pdfs/2008Proxy.pdf.