Experimentally Determining the Accuracy of a 3D Printed 80 Sappho Asteroid Model

Adam Stenson, Morgan Pearson*

Fort Hays State University, Kansas Academy of Mathematics and Science
AdamStenson04@gmail.com

Abstract

Through light curve inversion techniques three dimensional models have been created for 381 asteroids. The purpose of this experiment is to determine the accuracy of the model for one of these asteroids, 80 Sappho. To determine the accuracy the control light curve, acquired from the Minor Planet Center, will be compared to the experimental light curve. Using a 3D printed model of 80 Sappho the experimental light curve is created by simulating the Sun, Earth, and the rotation of the asteroid inside a dark box. The control and experimental light curves will then be compared using plotting software.

I. Introduction

The Database of Asteroid Models from Integration Techniques, which will be referred to as DAMIT from this point, is a collection of 3D models created from each asteroid light curves. There are currently 648 models for 381 asteroids.

I. Lightcurve Inversion Techniques

DAMIT was developed by Josef Ďurech, Mikko Kaasalainen, and Vojtěch Sidorin. They are based at Charles University in Prague, Tampere University of Technology in Finland, and the Astronomical Institute of the Academy of Science of the Czech Republic in Prague, respectively [1]. The basis for DAMIT and the inversion techniques is to create 3-dimensional models of asteroids that have not been directly imaged. Due to the fact that most asteroids are too small to be directly imaged many astronomers, amateur and professional, have collected thousands of light curves. One collection of these light curves, created by the International Astronomical Union's Minor Planet Center, contains over 1.5 million observations for 2350+ objects. The inversion techniques described [1] use these to observed light curves to create the models in DAMIT.

II. 80 Sappho

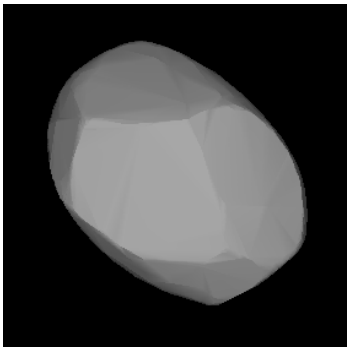


Figure 1: Computer Rendering of 80 Sappho Model

The asteroid studied, 80 Sappho, is a main belt asteroid that was discovered in 1864 [2]. Physical and orbital elements of the asteroid, from the Jet Propulsion Laboratory Small Body

*A special thanks to Dr. Jack Maseberg, the Science and Mathematics Education Institute, the Kansas Academy of Mathematics and Science, the FHSU MakerSpace, and our research advisor, Dr. Paul Adams.

Database, are shown in Table 1.

Characteristic	Measured Value
Inclination	8.664°
Rotational Period	14.03 hours
Orbital Period	3.48 years
Diameter	78.39 km
Semi-Major Axis	2.296 AU
Absolute Magnitude	7.98 mag

Table 1: Physical and Orbital Characteristics of 80 Sappho

II. Materials & Methods

I. Materials List

1. 8 ft^3 dark box built from 2 inch polystyrene insulating foam.
2. MakerBot Replicator 2 3-D printer with white PLA plastic.
3. A Feit Electric 500 Lumen LED flashlight
4. 500 to 1000 nm wavelength phototransistor
5. 22 gauge multi-color wires
6. 5.6 kΩ resistor
7. 0.22 μC capacitor
8. Arduino UNO R3
9. Unipolar Stepper Motor
10. Stepper Motor Controller
11. An integrated circuit
12. A breadboard

II. Arduino Setup

The Arduino UNO R3 was wired to a standard solderless breadboard via color coded wiring. The power supply came from two 12V AC adapters that were also wired into the breadboard.

The pins used for direction of the stepper motor and steps the stepper motor took were pins 2 and 3 (outputs). The stepper motor was coded to make one full rotation in the clockwise direction in 384 steps. There was a delay time of 6400 microseconds which determined the speed at which the stepper motor would rotate the asteroid.

The phototransistor that recorded the light intensity was wired directly into the breadboard in relation with the resistor. The analog input used was A0 and the analog output pin used was pin 9. The input value for the light and the stepper motor steps were concurrent. As the asteroid was rotating, the light sensor was coded to record a value at a speed of 300 milliseconds.

III. Data Collection

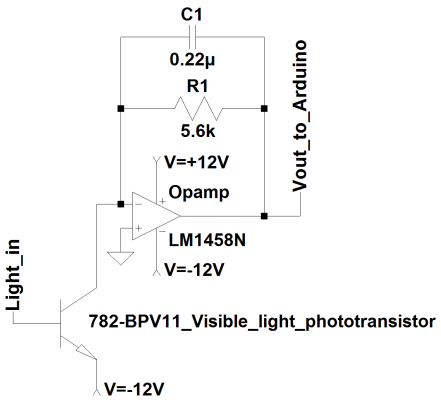
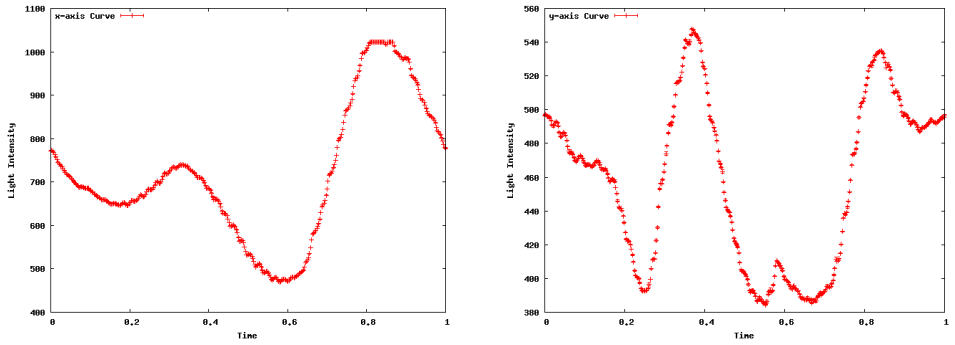


Figure 2: Arduino Circuit Diagram

The Arduino had a limited resolution, only being able to record values between 0 and 1023. The values were recorded on units of voltage, which is proportional to intensity. The Arduino could be further calibrated to record in intensities with units of watts per meters squared. The Arduino code for the data collection can be found in Appendix A.2.

The data was collected in four component curves, one for the the x, y, and z axes as well as a control. The control is the entire experimental setup without the asteroid allowing the removal of noise from the stepper motor and any surrounding surfaces that might re-



(a) x axis (b) y axis

(c) z axis (d) Control & I_{ref}

Figure 3: Componant curves from Test Run 1

flect light into the phototransistor. Each component curve was recorded 10 times so that any random error can be accounted for. The component curves from the first test run are shown in Fig. 3.

IV. Data Analysis

The data from the 10 trials of each component curve are averaged together and the standard error is calculated. The four component curves are then combined using the principle of superposition. The three axes are added together while the control curve is subtracted. The statistical error from each component curve is then propagated to the composite curve. Using Eq. (1) the intensities values in the composite curve are converted to magnitude. The control curve was used for I_{ref}.

$$m = -2.5 \log_{10} \left(\frac{I_1}{I_{ref}} \right) \tag{1}$$

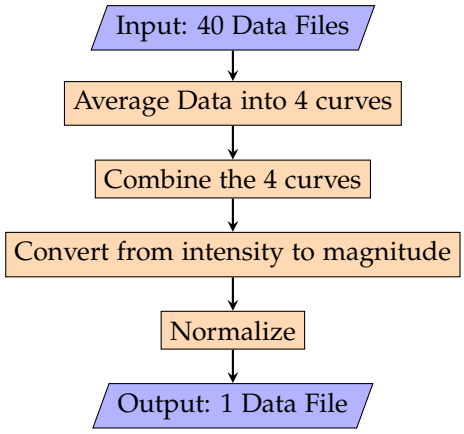
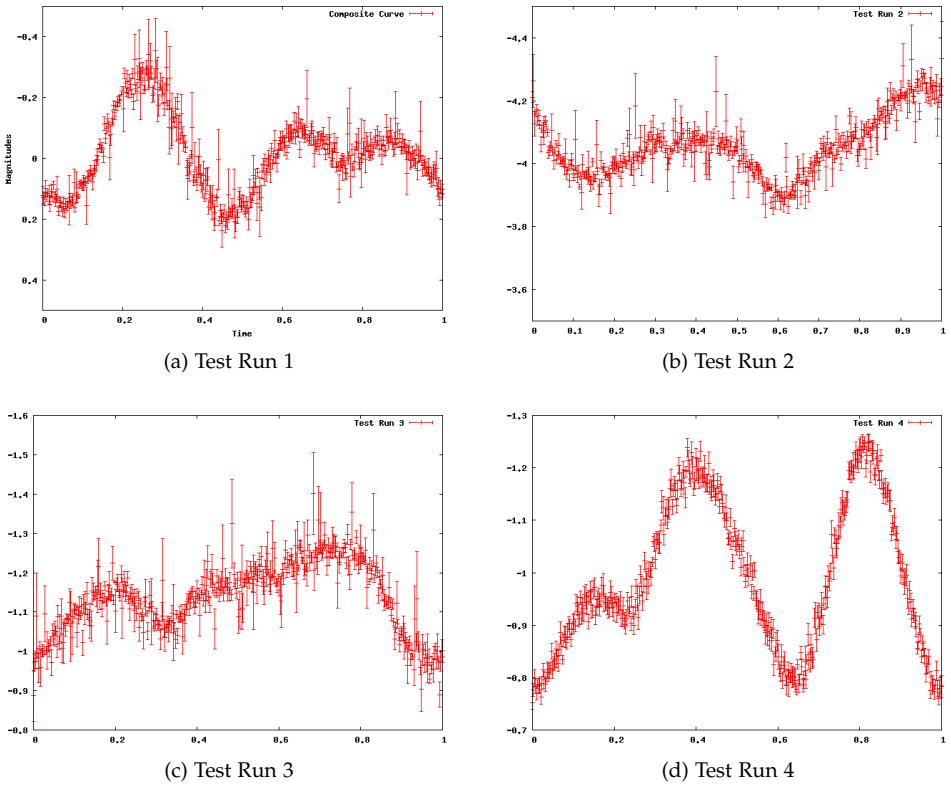


Figure 4: Data Analysis Process



(a) Test Run 1

(b) Test Run 2

(c) Test Run 3

(d) Test Run 4

Figure 5: Results of Test Runs 1-4

The data was then normalized. The full analysis process was automated using the C++ code in Appendix A.1 and is illustrated in Fig. 4. The entire process starts with 40 data files, one for each of the 10 trials for each of the 4 component curves in units proportional to intensity, and ends with one composite curve in units of apparent magnitude.

V. Error Determination

When averaging the 10 trials of each component curve together the statistical error. Equation (2) is used to do this where σ is the standard deviation and N is the number of trials.

$$\Delta I = \frac{\sigma}{\sqrt{N}} \tag{2}$$

The calculated statistical errors were propagated through the rest of the calculations.

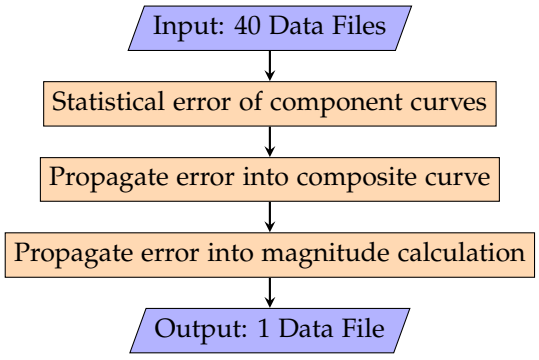
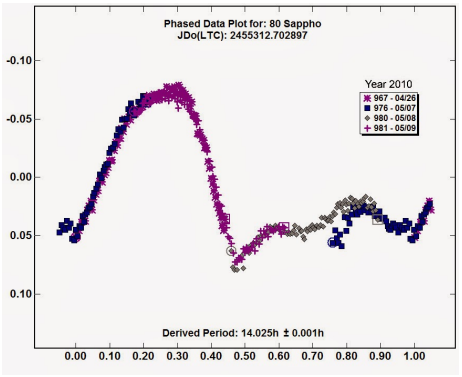


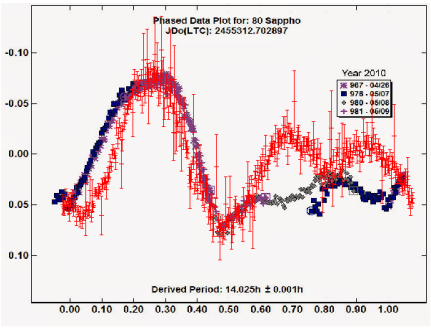
Figure 6: Error Calculations

$$\begin{aligned} \bar{x} &= f(\bar{a}, \bar{b}) \\ \Delta x_a &= f(\bar{a} \pm \Delta a, \bar{b}) - \bar{x} \\ \Delta x_b &= f(\bar{a}, \bar{b} \pm \Delta b) - \bar{x} \\ \Delta x &= \sqrt{(\Delta x_a)^2 + (\Delta x_b)^2} \end{aligned} \tag{3}$$

The error propagation method used is shown in Eq. (3) and the full error determination pro-



(a) Minor Planet Center Light Curve for 80 Sappho



(b) Overlay of Test Run 1 and the Light Curve from the Minor Planet Center

Figure 7: Results of Test Run 1

cess is illustrated in Fig. 6.

III. Results

The resultant composite curves are compared to the curve from the Minor Planet Center shown in Fig. 7(a). Test Run 1, Fig. 5(a), has a similar shape which is shown in Fig. 7(b). However, the three other test runs are not visually comparable. It is possible that phase shifts in the component curves caused the variances in Test Runs 2 through 4. This could have been easily avoided by indexing the asteroid at each axis with the stepper motor so that all the tests are in phase.

IV. Discussion

I. Nutations, Precession, and Translation

The rotation model used in this experiment is greatly simplified. Aside from the rotation about the three axes represented in this experiment, 80 Sappho also has nutations, precession, and translation components to its light curve.

Nutations are the oscillations of the asteroid's axes.

Precession is the change in the direction of the asteroid's axes of rotation over time.

Translation is the motion of the asteroid around the Sun.

These three aspects of the asteroid's rotation could have a substantial impact in the light curve. Nutations and the precession change the asteroid's orientation about its axes to the observation point. Translation changes the angle between the asteroid and the observation point. Finding a way to include these aspects of the rotation into the experiment would allow for a more accurate curve.

II. Phase Angles

The phase angle of the collected component curves could be another explanation for the variance in composite curves over the four test runs. This could be easily prevented by indexing the asteroid with the stepper motor so that the placement is the same for all trials. This phase shift can also be corrected during the data analysis process. In the case of this experiment the component curves of the first test run can be defined to be at a phase angle of 0 radians. The component curves of the remaining test runs could then e brought into phase with the first test run.

III. Arduino & Stepper Motor

Improvements within this experiment could be made regarding the Arduino set-up. Using

a bipolar motor instead of a unipolar motor more steps could be made within one full rotation.

References

[1] J. Ďurech, V. Sidorin, and M. Kaasalainen, *Damit: a database of asteroid models*, Astronomy & Astrophysics **513** (2010), no. A46.

[2] Jet Propulsion Laboratory, *Small body database: 80 sappho*.

A. Code Used in the Experiment

I. Data Reduction and Normalization

```
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

void avg(ifstream& input1, ifstream& input2, ifstream& input3, ifstream& input4, ifstream
    & input5, ifstream& input6, ifstream& input7, ifstream& input8, ifstream& input9,
    ifstream& input10, ofstream& output);

double stddev(double data1, double data2, double data3, double data4, double data5,
    double data6, double data7, double data8, double data9, double data10);

void composite(ifstream& inputx, ifstream& inputy, ifstream& inputz, ifstream& inputc,
    ofstream& output);

void mag(ifstream& composite, ifstream& control, ofstream& output);

int main()
{
    // finding the average of the x-axis trials
    ifstream xinput1("asteroids_x_test1.txt");
    ifstream xinput2("asteroids_x_test2.txt");
    ifstream xinput3("asteroids_x_test3.txt");
    ifstream xinput4("asteroids_x_test4.txt");
    ifstream xinput5("asteroids_x_test5.txt");
    ifstream xinput6("asteroids_x_test6.txt");
    ifstream xinput7("asteroids_x_test7.txt");
    ifstream xinput8("asteroids_x_test8.txt");
    ifstream xinput9("asteroids_x_test9.txt");
    ifstream xinput10("asteroids_x_test10.txt");
    ofstream xoutput("x_avg.txt");
    avg(xinput1, xinput2, xinput3, xinput4, xinput5, xinput6, xinput7, xinput8, xinput9,
    xinput10, xoutput);
    xinput1.close();
    xinput2.close();
    xinput3.close();
    xinput4.close();
    xinput5.close();
    xinput6.close();
    xinput7.close();
    xinput8.close();
    xinput9.close();
    xinput10.close();
    xoutput.close();
    // finding the average of the y-axis trials
    ifstream yinput1("asteroids_y_test1.txt");
    ifstream yinput2("asteroids_y_test2.txt");
    ifstream yinput3("asteroids_y_test3.txt");
    ifstream yinput4("asteroids_y_test4.txt");
    ifstream yinput5("asteroids_y_test5.txt");
    ifstream yinput6("asteroids_y_test6.txt");
    ifstream yinput7("asteroids_y_test7.txt");
    ifstream yinput8("asteroids_y_test8.txt");
    ifstream yinput9("asteroids_y_test9.txt");
    ifstream yinput10("asteroids_y_test10.txt");
```

```
    ofstream youtput("y_avg.txt");
    avg(yinput1, yinput2, yinput3, yinput4, yinput5, yinput6, yinput7, yinput8, yinput9,
    yinput10, youtput);
    yinput1.close();
    yinput2.close();
    yinput3.close();
    yinput4.close();
    yinput5.close();
    yinput6.close();
    yinput7.close();
    yinput8.close();
    yinput9.close();
    yinput10.close();
    youtput.close();
    // finding the average of the z-axis trials
    ifstream zinput1("asteroids_z_test1.txt");
    ifstream zinput2("asteroids_z_test2.txt");
    ifstream zinput3("asteroids_z_test3.txt");
    ifstream zinput4("asteroids_z_test4.txt");
    ifstream zinput5("asteroids_z_test5.txt");
    ifstream zinput6("asteroids_z_test6.txt");
    ifstream zinput7("asteroids_z_test7.txt");
    ifstream zinput8("asteroids_z_test8.txt");
    ifstream zinput9("asteroids_z_test9.txt");
    ifstream zinput10("asteroids_z_test10.txt");
    ofstream zoutput("z_avg.txt");
    avg(zinput1, zinput2, zinput3, zinput4, zinput5, zinput6, zinput7, zinput8, zinput9,
    zinput10, zoutput);
    zinput1.close();
    zinput2.close();
    zinput3.close();
    zinput4.close();
    zinput5.close();
    zinput6.close();
    zinput7.close();
    zinput8.close();
    zinput9.close();
    zinput10.close();
    zoutput.close();
    // finding the average of the control trials
    ifstream cinput1("I_ref_test1.txt");
    ifstream cinput2("I_ref_test2.txt");
    ifstream cinput3("I_ref_test3.txt");
    ifstream cinput4("I_ref_test4.txt");
    ifstream cinput5("I_ref_test5.txt");
    ifstream cinput6("I_ref_test6.txt");
    ifstream cinput7("I_ref_test7.txt");
    ifstream cinput8("I_ref_test8.txt");
    ifstream cinput9("I_ref_test9.txt");
    ifstream cinput10("I_ref_test10.txt");
    ofstream coutput("c_avg.txt");
    avg(cinput1, cinput2, cinput3, cinput4, cinput5, cinput6, cinput7, cinput8, cinput9,
    cinput10, coutput);
    cinput1.close();
    cinput2.close();
    cinput3.close();
    cinput4.close();
    cinput5.close();
    cinput6.close();
    cinput7.close();
    cinput8.close();
```

```
    cinput9.close();
    cinput10.close();
    coutput.close();
    // create the composite curve
    ifstream inputx("x_avg.txt");
    ifstream inputy("y_avg.txt");
    ifstream inputz("z_avg.txt");
    ifstream inputc("c_avg.txt");
    ofstream output("composite_intensities.txt");
    composite(inputx, inputy, inputz, inputc, output);
    inputx.close();
    inputy.close();
    inputz.close();
    inputc.close();
    output.close();
    // convert the intensities of the composite curve to magnitudes
    ifstream composite("composite_intensities.txt");
    ifstream control("c_avg.txt");
    ofstream output_mag("composite.txt");
    mag(composite, control, output_mag);
    composite.close();
    control.close();
    output_mag.close();
}

void avg(ifstream& input1, ifstream& input2, ifstream& input3, ifstream& input4, ifstream
    & input5, ifstream& input6, ifstream& input7, ifstream& input8, ifstream& input9,
    ifstream& input10, ofstream& output) {
    double steps[400], time[400], data[10][400], avg[400], err[400];
    int count = 0;
    while (!input1.eof() && !input2.eof() && !input3.eof() && !input4.eof() && !input5.
    eof() && !input6.eof() && !input7.eof() && !input8.eof() && !input9.eof() && !input10
    .eof()) {
        input1 >> steps[count] >> data[0][count];
        input2 >> steps[count] >> data[1][count];
        input3 >> steps[count] >> data[2][count];
        input4 >> steps[count] >> data[3][count];
        input5 >> steps[count] >> data[4][count];
        input6 >> steps[count] >> data[5][count];
        input7 >> steps[count] >> data[6][count];
        input8 >> steps[count] >> data[7][count];
        input9 >> steps[count] >> data[8][count];
        input10 >> steps[count] >> data[9][count];
        count++;
    }
    for (int i = 0; i < count; i++) {
        time[i] = 1-(((14.025/383) * steps[i])/14.025);
    }
    for (int a = 0; a < count; a++) {
        double total = data[0][a] + data[1][a] + data[2][a] + data[3][a] + data[4][a] +
    data[5][a] + data[6][a] + data[7][a] + data[8][a] + data[9][a];
        avg[a] = total / 10;
    }
    for (int j = 0; j < count; j++) {
        err[j] = stddev(data[0][j], data[1][j], data[2][j], data[3][j], data[4][j], data
    [5][j], data[6][j], data[7][j], data[8][j], data[9][j]) / sqrt(10);
    }
    for (int k = 0; k < count; k++) {
        output << time[k] << "     " << avg[k] << "      " << err[k] << endl;
    }
}
```

```
165
double stddev(double data1, double data2, double data3, double data4, double data5,
    double data6, double data7, double data8, double data9, double data10){
167     double stddev;
        int count = 10;
169     double data[10];
        data[0] = data1;
171     data[1] = data2;
        data[2] = data3;
173     data[3] = data4;
        data[4] = data5;
175     data[5] = data6;
        data[6] = data7;
177     data[7] = data8;
        data[8] = data9;
179     data[9] = data10;
        double numerator = 0;
181     double denominator = count;
        double total = 0;
183     for (int i = 0; i < count; i++) {
            total = total + data[i];
185     }
        double xbar = total/count;
187     for (int j = 0; j < count; j++) {
            numerator = numerator + pow((data[j] - xbar), 2);
189     }
        stddev = sqrt(numerator/denominator);
191     return stddev;
}
193
void composite(ifstream& inputx, ifstream& inputy, ifstream& inputz, ifstream& inputc,
    ofstream& output){
195     double time[400], xdata[400], ydata[400], zdata[400], cdata[400], xerr[400], yerr
    [400], zerr[400], cerr[400], err[400], data[400], derr1[400], derr2[400], derr3[400],
     derr4[400];
        int count = 0;
197     while (!inputx.eof() && !inputy.eof() && !inputz.eof()) {
            inputx >> time[count] >> xdata[count] >> xerr[count];
199         inputy >> time[count] >> ydata[count] >> yerr[count];
            inputz >> time[count] >> zdata[count] >> zerr[count];
201         inputc >> time[count] >> cdata[count] >> cerr[count];
            count++;
203     }
        for (int i = 0; i < count; i++) {
205         data[i] = xdata[i] + ydata[i] + zdata[i] - cdata[i];
        }
207     for (int j = 0; j < count; j++) {
            derr1[j] = (xdata[j] + xerr[j] + ydata[j] + zdata[j] - cdata[j]) - data[j];
209         derr2[j] = (xdata[j] + ydata[j] + yerr[j] + zdata[j] - cdata[j]) - data[j];
            derr3[j] = (xdata[j] + ydata[j] + zdata[j] + zerr[j] - cdata[j]) - data[j];
211         derr4[j] = (xdata[j] + ydata[j] + zdata[j] - cdata[j] - cerr[j]) - data[j];
            err[j] = sqrt( pow(derr1[j], 2) + pow(derr2[j], 2) + pow(derr3[j], 2) + pow(derr4
    [j], 2) );
213     }
        for (int k = 0; k < count; k++) {
215         output << time[k] << "     " << data[k] << "      " << err[k] << endl;
        }
217 }

219 void mag(ifstream& composite, ifstream& control, ofstream& output){
```

```
    double m[400], err[400], derr1[400], derr2[400], time[400], data1[400], data2[400],
    err1[400], err2[400];
    int count = 0;
    while (!composite.eof() && !control.eof()) {
        composite >> time[count] >> data1[count] >> err1[count];
        control >> time[count] >> data2[count] >> err2[count];
        count++;
    }
    for (int i = 0; i < count; i++) {
        m[i] = -2.5*log10(data1[i]/data2[i]);
        derr1[i] = -2.5*log10( (data1[i] + err1[i])/data2[i]) - m[i];
        derr2[i] = -2.5*log10( data1[i]/(data2[i] + err2[i])) - m[i];
        err[i] = sqrt( pow(derr1[i], 2) + pow(derr2[i], 2) );
        output << time[i] << "    " << m[i] << "    " << err[i] << endl;
    }
}
```

II. Arduino Code

```
int analogInPin = A0;  // Analog input pin that the potentiometer is attached to
int DIR = 3;                          // PIN  3 = DIR
int STEP = 2;                         // PIN  2 = STEP
int MS1 = 13;                         // PIN 13 = MS
int MS2 = 9;                          // PIN  9 = MS2
int SLEEP = 12;                       // PIN 12 = SLP
int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
  pinMode(DIR, OUTPUT);               // set pin 3 to output
  pinMode(STEP, OUTPUT);              // set pin 2 to output
  pinMode(MS1, OUTPUT);               // set pin 13 to output
  pinMode(MS2, OUTPUT);               // set pin 9 to output
  pinMode(SLEEP, OUTPUT);             // set pin 12 to output
  digitalWrite(SLEEP, HIGH);          // Set the Sleep mode to AWAKE.
  digitalWrite(DIR, LOW);             // Set the direction change LOW to HIGH to go in
    opposite direction
  digitalWrite(MS1, HIGH);            // Do this for 1/8 step mode
  digitalWrite(MS2, HIGH);            // Do this for 1/8 step mode
}

void loop() {

    int i = 0;
    int j = 0;
    while(i<384)                      // One full rotation is 384 steps...
    {
      digitalWrite(STEP, LOW);        // This LOW to HIGH change is what creates the..
      digitalWrite(STEP, HIGH);       // .."Rising Edge" so the easydriver knows to when
    to step.
      delayMicroseconds(6400);        // This delay time determines the speed of the
    stepper motor (was 1600, 800, 400, or 200 microseconds)


    delay(2); //wait for adc to settle
```

```
    // read the analog in value:
37  sensorValue = analogRead(analogInPin);
    // print the results to the serial monitor:
39  Serial.print(i);
    Serial.print("     ");
41  Serial.println(sensorValue);
    i++;
43  }
    delay(1000);
45 }
```