Exercise 1: measurement, uncertainty, graphing, & propagation of error

Purpose: to become familiar with common data taking and analysis techniques.

Introduction

Every measurement is associated with some degree of uncertainty. Generally, Δq is used to indicate the numerical uncertainty of a physical quantity q, which can be either a measurement or a calculated quantity. Note that the symbol we are using for the uncertainty may also be used to signify a change in a quantity, which should not be confused with it being used as uncertainty. If we measure x to be 20.6 cm with an uncertainty $\Delta x = \pm$ 0.2 cm, this means that the value lies in the range of 20.4 cm to 20.8 cm. Any calculation involving one or more measurements (which have uncertainties) also has some degree of uncertainty. The process of deriving an expression to calculate this uncertainty is called the *propagation of uncertainty*.

Errors in the data-collection process can be either *random* or *systematic*. Random errors arise due to slightly different measuring conditions and to different observer interpretations. Systematic errors occur when experimental parameters that effect the measurement are not taken into consideration. Systematic errors are reproducible, and are typically compensated in the experiment or analysis. One way to find the random error in a particular measurement is to measure the same quantity many times in the same manner. The higher the number of trials, the closer the average gets to the true value (assuming no systematic errors and a normal distribution of errors).

Uncertainty in measured quantities

A. Standard deviation of multiple measurements

The corrected standard deviation σ_x of a set of values x is a measure of the average variation of individual measurements from a mean value when assuming a normal distribution. The standard deviation of a physical parameter associates a single measurement's variation from the true value for a specific set of experimental conditions. The corrected standard deviation of a measured value x for a discrete set of N measurements is given by

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{N} (x_i - x_{\text{avg}})^2}{N-1}}$$

where

$$x_{\text{avg}} = \frac{1}{N} \sum_{i}^{N} x_i$$

As an activity, take a ball and raise it to eye level. Use the stop watch to measure the time it takes the ball to drop. Do this measurement five times dropping from the same height for each drop. Measure the height at which you are consistently dropping the ball. Make a table similar to Table 1 to record the data. Also calculate the 1) average value and 2) corrected standard deviation and place them in the table.

Table 1: Measurements of the time it takes for the ball to hit the floor

Measurement	1	2	3	4	5	Average	Std. dev.
Time (s)							

$h =$ _________________m $\Delta h =$ _________________m

(height) (uncertainty in height)

B. Estimated uncertainty of a measured value

In some instances, multiple measurements give the same value (or nearly the same value) for all measurements. This does not mean that the value is exact. Instead, the precision of the measuring apparatus is less than the accuracy of the values measured. Thus, the accuracy is greater than the precision, which means that the primary source of random error is due to the precision of the measuring apparatus. Note that parallax, placement of the instrument, and irregularities commonly add up to the estimated uncertainty. Note that systematic errors can occur for biased measurements. 1) Use the caliper to measure the diameter of the ball, and 2) estimate the uncertainty in the measurement. 3) Also, estimate the uncertainty of the height from when you dropped the ball.

Propagation of uncertainty

The most common method of propagating uncertainty is based on the Taylor series expansion of a function $f(x)$. If we know a function $f(x)$ at some given value x_0, then we can approximate the value of the function at $x = x_0 \pm \Delta x$ by expanding the function around x_0,

$$f(x_0 \pm \Delta x) \approx f(x_0) + \sum_{n=1}^{\infty} \frac{1}{n!} \frac{d^n}{dx^n} f(x) \bigg|_{x=x_0} (\pm \Delta x)^n .$$

When the uncertainty is relatively small (as we expect for any value worth reporting), we may truncate the Taylor series expansion to first-order,

$$f(x_0 \pm \Delta x) \approx \boxed{f(x_0)} + \boxed{\frac{d}{dx} f(x) \bigg|_{x=x_0} (\pm \Delta x) .}$$

(average value, f) (± uncertainty, Δf)

The value and propagated uncertainty in the function may be written as $f(x_0) \pm |f(x_0 \pm \Delta x)|$, where $f(x_0 \pm \Delta x) = \Delta f$.

For a function that depends on multiple independent variables, $f(x, t, q)$, we must approximate the function for a change in a single variable while holding all other variables constant. Thus, using the linear approximation for the example function $f(x, t, q)$, we have the following equations

$$f(x_0 \pm \Delta x, t_0, q_0) \approx f(x_0, t_0, q_0) + \frac{\partial}{\partial x} f(x, t_0, q_0)\bigg|_{x=x_0} (\pm \Delta x)$$
$$f(x_0, t_0 \pm \Delta t, q_0) \approx f(x_0, t_0, q_0) + \frac{\partial}{\partial t} f(x_0, t, q_0)\bigg|_{t=t_0} (\pm \Delta t)$$
$$f(x_0, t_0, q_0 \pm \Delta q) \approx f(x_0, t_0, q_0) + \frac{\partial}{\partial q} f(x_0, t_0, q)\bigg|_{q=q_0} (\pm \Delta q).$$

Because the function is evaluated for all independent variables, the propagated uncertainty may be found by using the Pythagorean theorem,

$$\Delta f = \sqrt{\left[\frac{\partial}{\partial x} f(x, t_0, q_0)\bigg|_{x=x_0} (\Delta x)\right]^2 + \left[\frac{\partial}{\partial t} f(x_0, t, q_0)\bigg|_{t=t_0} (\Delta t)\right]^2 + \left[\frac{\partial}{\partial q} f(x_0, t_0, q)\bigg|_{q=q_0} (\Delta q)\right]^2}.$$

For free fall of an object starting from rest, the position as a function of time is given by

$$x = \frac{1}{2} g t^2.$$

Solve the above equation for g. Derive an expression for the uncertainty Δg from the uncertainties Δx and Δt. Calculate the average value of g and the uncertainty Δg from the dropped ball data you entered in Table 1 (you only need to use the values from the eye level drop). Make sure and write them down as $g \pm \Delta g$ with units. For example taking $f_{\text{avg}} =$ 1.2 m/s^2 and $\Delta f =$ 0.1 m/s^2, we would write $f =$ (1.2 ± 0.1) m/s^2.

Linear least-squares method

Sometimes we want to verify the relationship between parameters. As an example, suppose we want to verify the relationship discovered by Newton that the acceleration a of an object is equal to the total force applied F divided by the mass of the object m. Symbolically, this may be written simply as

$$a = \frac{F}{m}.$$

It becomes clear quite rapidly that the "total force applied" might not be the force that we are applying, F_{app} in our experiment, *i.e*. other forces may be present such as friction. Fortunately, we may separate these forces, which gives

$$a = \frac{F_{\text{app}}}{m} + \frac{F_{\text{other}}}{m}.$$

Suppose we measured the values and uncertainties of the applied force and acceleration, which are given in Table 2. We cannot simply find the mass by using the equation $F = ma$ because we do not know the value of the "other" forces. If F_{other} is constant over the range of experimental parameters listed in Table 2, then we may remove the dependence on F_{other} by taking the difference between points.

Table 2: Applied force and acceleration data

F_{app} (N)	2.1±0.1	3.9±0.2	6.0±0.1	8.2±0.1	9.9±0.2
a (m/s^2)	0.90±0.08	1.8±0.2	3.1±0.1	4.0±0.2	5.2±0.1

Because there are many points, there are many combinations of differences that can lead to small variations in the slope of the linear relationship. A method to find the best fit line for this type of system is called the linear least-squares method. Thus, our goal is to find the slope and the intercept of the equation $F_{\text{app}} = ma - F_{\text{other}}$. This method can be simplified greatly by only taking the graphical trendline from the mean values, where we assume that the effects from the data point uncertainties will be negligible.

To complete a linear least-squares fit with multiple data points, it is best to use a computer to perform the calculation. We will arrive at a simple C/C++ program (If preferred, a python version is also provided at the end of this laboratory assignment) by the end of this section. This program can be used to calculate the a) slope, b) uncertainty in the slope, c) intercept, and d) uncertainty in the intercept. As a preliminary, we write the following code: (comments are given between the starting /* and ending */)

```
#include <iostream> /* if compiled in C, change to stdio.h */
#include <cmath>    /* if compiled in C, change to math.h for pow function */

/* list subroutines in this code */
double sumarr(double data[], int N);
void printvals(double vals[]);

/* Begin main program, returns 0, null argument */
int main(void)
{
   /* declare constant integers */
   const int N = 5;

   /* declare integer variables */
   int i;

   /* declare double precision variables */
   double xs,ys,xxs,xys,sxx,sxy,slp,intrcpt,linsum,sigysqr;
   double slpuncert,intrcptuncert;

   /* declare double precision arrays */
   double xx[N],xy[N],vals[4],yhat[N],xdata[N],ydata[N],resid[N];

   /* fill in mean acceleration data from Table 2 */
   xdata[0] = 0.9;
   xdata[1] = 1.8;
   xdata[2] = 3.1;
   xdata[3] = 4.0;
   xdata[4] = 5.2;
```

```
/* fill in mean force data from Table 2 */
ydata[0] = 2.1;
ydata[1] = 3.9;
ydata[2] = 6.0;
ydata[3] = 8.2;
ydata[4] = 9.9;
```

After the initial setup, we begin by summing the array elements for the independent and dependent variable data.

```
/* sum data array elements using sumarr subroutine below */
xs = sumarr(xdata,N);
ys = sumarr(ydata,N);
```

Next we square each data point for the independent variable, and also multiply the each data point's independent and dependent variable values.

```
/* multiply xdata by ydata, also square xdata elements */
for (i=0; i<N; i++)
{
   xx[i]= pow(xdata[i],2);
   xy[i]= xdata[i]*ydata[i];
}
```

Then we sum each element of those arrays.

```
/* sum elements of the products and squares arrays */
xys = sumarr(xy,N);
xxs = sumarr(xx,N);
```

Next we calculate the corrected sum-of-squares and the corrected sum-of-products from the above values.

```
/* calculate the corr SOS and corr SOP */
sxy = xys - xs*ys/N;
sxx = xxs - pow(xs,2)/N;
```

We are now able to determine the slope and intercept of the best fit line.

```
/* find the slope and the intercept */
slp = sxy/sxx;
intrcpt = ys/N - slp*xs/N;
```

We are not quite done yet as we would also like to know the uncertainty in the slope and intercept values. Thus, we need to calculate the residual y values based on the original data and fit parameters.

```
/* calculate residual y values from data and fit params */
for (i=0; i<N; i++)
{
   resid[i]= pow(ydata[i] - intrcpt - slp*xdata[i],2);
}
```

Then we determine the uncertainty σ_y^2.

```
    /* calculate sigma_y squared */
    linsum = sumarr(resid,N);
    sigysqr = linsum/(N-2);
```

Finally we estimate the uncertainty in both the slope and the intercept.

```
    /* estimate the uncertainty in the slope and intercept */
    slpuncert = sqrt(sigysqr*N/(N*xxs - pow(xs,2)));
    intrcptuncert = sqrt(sigysqr*xxs/(N*xxs - pow(xs,2)));
```

We fill in an array to pass along to the *print* subroutine

```
    /* set array values to pass to the printvals subroutine */
    vals[0] = slp;
    vals[1] = slpuncert;
    vals[2] = intrcpt;
    vals[3] = intrcptuncert;
```

We call the subroutine to display the estimated values and then close out the main function

```
    /* call subroutine to print the estimated values */
    printvals(vals);
    return 0;
}
```

Below the main program, we code in the array summation subroutine

```
double sumarr(double data[],int N)
{
    int i;
    double sum = 0.0;
    for (i=0; i<N; i++)
    {
        sum = sum + data[i];
    }
    return(sum);
}
```

Lastly, we add the subroutine to print the estimated values.

```
void printvals(double vals[])
{
    printf("The slope is %lf +/- %lf\n",vals[0],vals[1]);
    printf("The intercept is %lf +/- %lf\n",vals[2],vals[3]);
}
```

Note that the C/C++ languages need to be compiled. If you do not have a computer with your favorite compiler on it yet, then you can write this code in a text editor such as notepad, where you may copy and paste it into an online test compiler such as https://www.ideone.com . This code was tested in C++14 and is working properly.

Determine the best fit parameters (and their uncertainties) for the mean values given in Table 2. Determine whether F_{other} is acting with or against the applied force.

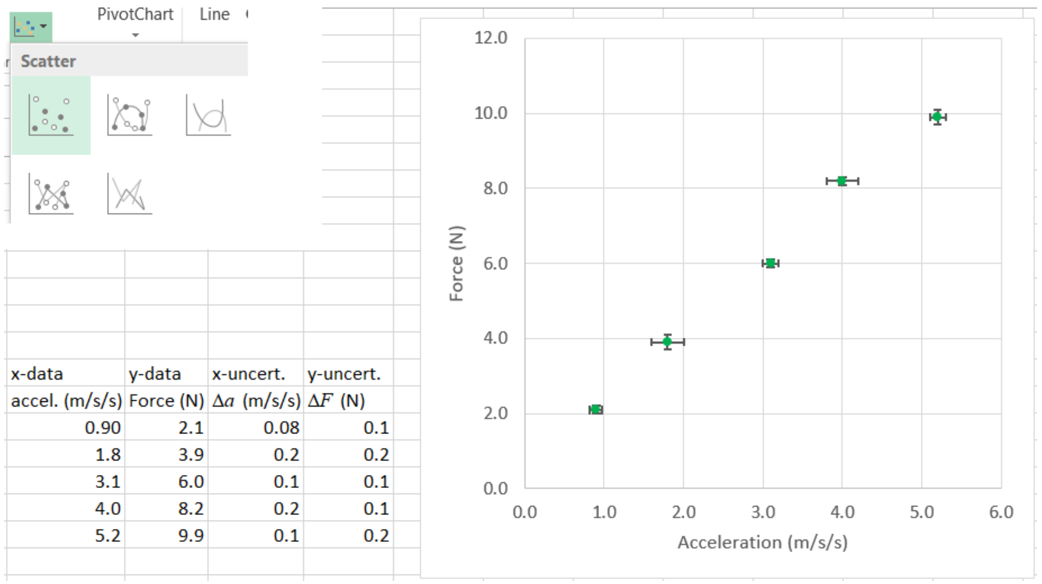
<u>Graphing</u>

Two-dimensional graphs display information of a dependent variable based on the value of an independent variable. For a continuous function $y(x)$, y is called the dependent variable and x is called the independent variable. Thus, the value of y depends on the value of x. The dependent variable's axis is oriented in the vertical direction, and the independent variable's axis is oriented in the horizontal direction. If these variables have units, then they must also be clearly displayed after the axes labels.

When plotting data, the mean values should first be placed in columns with every dependent variable value corresponding to the adjacent independent variable value. For example, Microsoft Excel plots two column data by default when the x-data is in the first column and the y-data is in the second column as shown in Figure 1 using the data from Table 2.

The uncertainties of each variable can be inserted as "error bars" in the graph. First, we make a column of x uncertainties corresponding to the average x-data points, and a column of y uncertainties corresponding to the average y-data points as shown in Figure 1. To make the graph in excel, we select the x and y columns presented in the order shown in Figure 1. Then in the insert tab, select the "scatter plot" option. You can use the "+" symbol at the top right of the graph when the graph is highlighted. You should select "axis titles" to label your plot's axes.

For the uncertainties, click on the "+" symbol and select error bars. The error bars that show up are NOT the error that you have in you two uncertainty columns. Thus, you must also click the small arrow to the right of the "error bars" selection and click "more options." To the right of your screen there is a "format error bars" window that pops up. At the bottom, select the "custom" radio button. Then click on the specify value bottom and a small window pops up. Use this window to insert your custom error bars into the graph (They will be the same values for the positive and negative error bars). Do custom error bars for both the x-error and y-error. The end result will be the graph in Figure 1.



x-data	y-data	x-uncert.	y-uncert.
accel. (m/s/s)	Force (N)	Δa (m/s/s)	ΔF (N)
0.90	2.1	0.08	0.1
1.8	3.9	0.2	0.2
3.1	6.0	0.1	0.1
4.0	8.2	0.2	0.1
5.2	9.9	0.1	0.2

Figure 1. Excel graph from the data given in Table 2 which includes axes labels and error bar representations of the data point uncertainties.

Take the ball and drop it from chest level 5 times recording the times and the height and fill in another table similar to Table 1 as you did from eye level. Drop the ball again from waist level 5 times and fill in another table. Do this same procedure once again for knee level. You should now have four data points for your height and four data points for the time, each with uncertainty estimates.

Table 3: Drop data

	$\frac{1}{2}t^2$	h
Knee-level		
Waist-level		
Chest-level		
Eye-level		

We know that the time it takes the ball to drop from rest is given by $x = (g/2)t^2$. Thus, to get a linear curve we should take the height as our dependent variable, and one-half the time squared as our independent variable. You should fill in a table similar to that shown in Table 3. Make sure that you propagated the error correctly for the $t^2/2$ term, where

$$\Delta\left(t^2\right) \neq (\Delta\, t)^2.$$

Use the propagation of uncertainty section as your guide to finding this uncertainty estimate. Graph h vs. $t^2/2$ making sure to add the error bars to display the uncertainty. Then, find the slope, uncertainty of the slope, intercept, and uncertainty in the intercept using your code (remember that $N =$ 4 for this analysis).

Appendix I: Matlab linear regression code

```
function linearregression
% Set up data columns
ydata = [2.1,3.9,6.0,8.2,9.9]';
xdata = [0.9,1.8,3.1,4.0,5.2]';
N = length(ydata);

%sum data array elements
xs = sum(xdata);
ys = sum(ydata);

%multiply xdata elements by ydata elements, also square
    xdata elements
xx = xdata.^2;
xy = xdata.*ydata;

%sum elements of the products
xys = sum(xy);
xxs = sum(xx);
```

```
%calculate the corr SOS and corr SOP
sxy = xys - xs*ys/N;
sxx = xxs - (xs^2)/N;

%find the slope and the intercept
slp = sxy/sxx;
intrcpt = ys/N - slp*xs/N;

%calculate residual y values from data and fit params
resid = (ydata - intrcpt - slp*xdata).^2;

%calculate sigma_y squared
linsum = sum(resid);
sigysqr = linsum/(N-2);

%estimate the uncertainty in the slope and intercept
slpuncert = sqrt(sigysqr*N/(N*xxs - xs^2));
intrcptuncert = sqrt(sigysqr*xxs/(N*xxs - xs^2));

%print values
disp(['The slope is ' , num2str(slp), ' +/- ', num2str(
    slpuncert)]);
disp(['The intercept is ' , num2str(intrcpt), ' +/- ',
    num2str(intrcptuncert)]);

end
```

This code was tested and works in current versions of Matlab.

Appendix II: Python linear regression code using Numerical Python (numpy)

```
import numpy as np

#setup data columns
ydata = np.array([2.1, 3.9, 6.0, 8.2, 9.9]);
xdata = np.array([0.9, 1.8, 3.1, 4.0, 5.2]);
N = len(ydata);

#sum data array elements
xs = np.sum(xdata);
ys = np.sum(ydata);

#multiply xdata elements by ydata elements, also square xdata elements
xx = xdata**2;
xy = xdata*ydata;

#sum elements of the products
xys = np.sum(xy);
xxs = np.sum(xx);
```

```
#calculate the corr SOS and corr SOP
sxy = xys - xs*ys/N;
sxx = xxs - (xs**2)/N;

#find the slope and the intercept
slp = sxy/sxx;
intrcpt = ys/N - slp*xs/N;

#calculate residual y values from data and fit parameters
resid = (ydata - intrcpt - slp*xdata)**2;

#calculate sigma_y squared
linsum = np.sum(resid);
sigysqr = linsum/(N-2);

#estimate the uncertainty in the slope and intercept
slpuncert = np.sqrt(sigysqr*N/(N*xxs - xs**2));
intrcptuncert = np.sqrt(sigysqr*xxs/(N*xxs - xs**2));

#print values
print("The slope is ", slp, " +/- ", slpuncert, ".")
print("The intercept is ", intrcpt, " +/- ", intrcptuncert, ".")
```

This code was tested and works in Python 3.6.