ezLecture: Data Analysis Tutorial

Our goal during this tutorial is to improve your data analysis skills. Improving these skills will give you a greater understanding of the data in research papers and the conclusions that are drawn from this data. If you want to understand the latest research breakthroughs and make breakthroughs of your own, you will need to be able to manipulate data using a powerful data analysis tool. For this tutorial, that tool will be the software program IGOR. Learning IGOR will require some effort on your part now, but will allow you to easily manipulate data in the future.

Let's say that we have just gotten some data and would like to graph the data and prepare it for publication. For instance let's say we have just obtained the data in the paper by Junker et al., "Ligand-Dependent Equilibrium Fluctuations of Single Calmodulin Molecules," *Science* 2009. I have extracted the data from Fig. 1B-C for you. Your job will be to analyze the data and reproduce Fig. 1B-C that appear in the publication. The data from Fig. 1B (bottom graph) is located in Junker_Fig1B.txt and is has a lower sampling rate, so it looks slightly different. The data from Fig. 1C is located in Junker_Fig1C.txt. You do not have to understand the paper in order to complete the tutorial.

For those that are interested, the following description about the data may be of use. If you are not interested, then skip this paragraph. The data we will be looking at is the unfolding of the two domains of calmodulin. To unfold the molecule, the two ends of the molecule are held and stretched at a constant rate of 1 nm/s. In the beginning, the two ends are pulled apart relatively easily, and the force remains low. In this regime, the entropy decreases as the molecule becomes more ordered and the ends pull apart. However, as the two ends get further apart and the extension of the molecule increases, the force suddenly increases quite rapidly as the bonds that hold the protein's tertiary structure together begin to be stretched. Eventually there is a "pop" as one of the domains of calmodulin is completely unfolded. During this "pop" the force suddenly decreases while the extension increases. To understand this unfolding, the authors have analyzed these events to figure out the unfolded length of each domain and the properties of the energy landscape. We will be following this data analysis as we learn to navigate IGOR.

Finally, during this tutorial you will be watching a series of videos dubbed "ezLectures". The number one goal of these ezLectures is to deliver a high amount of content in only 2-4 minutes. This means that you might be overwhelmed when watching them at first. My recommendation is to watch them once or twice before you start the assignment. Then, attempt to do the assigment by trial and error. Once you have completed the assignment, watch the video again. If you understand everything, you can move on to the next section.

Learning Goals:

1. Install IGOR and learn some basic commands.
2. Load data into IGOR and graph it.
3. Fit curves in IGOR.

4. Make and save figures using IGOR.
5. Use procedures in IGOR.

1. Install IGOR and learn some basic commands.

A) First, install IGOR. There is a trial version that you can download for free and use for a trial period.

B) Open IGOR and look around. You should notice some tabs at the top, a table in the work area, and a command window at the bottom. Let's learn how to navigate IGOR by trial and error. Try to figure out how to do each of the following tasks by either right clicking the table, using the tabs at the top, or by using the command window. You may use the IGOR help files or the internet for help if you get stuck.

- Go ahead and enter the following values: 2, 3.5, 5, 10, 11, 18, and 20 into a column in Table0. This will create an array of values that is called a "wave" in IGOR.
- Insert a point of 7.5 between the values 5 and 10 by using the right click on your mouse.
- Delete a point using the right click on your mouse.
- Rename the wave to be "mywave" without the quotes. Wave names should not have spaces, quotes, or any unusual characters, just letters, numbers, or the underscore character. To do this, right click the name of the wave in the table.
- Print the statistics of "mywave" in the command window. Find "Wave Stats" under the "Analysis" tab at the top.
- Duplicate "mywave". Call the duplicate wave: "mywave2". Look in the "Help" tab at the top and click on "Command Help". Scroll through the commands until you find "duplicate". This will give you instructions on how to enter the command in the command window (window where there is a running tab of actions that you have completed).
- Append "mywave2" to Table0. Click on Table0 and use the "Table" tab at the top.
- Remove "mywave" and "mywave2" from Table0. Right click on the wave names in the table.
- Open a new table, Table1, and then add "mywave" and "mywave2" to this table. Use the "Windows" tab at the top to add the table.
- Kill "mywave2" so that it is completely deleted. Then duplicate "mywave" again to remake "mywave2". Right click on the wave name in the table or use the "Data" tab at the top.

C) Watch the ezLecture: Command Line in Igor. Then do the following tasks.

- Type in the command window: "mywave2=4" without the quotes. You should notice that every value in the wave is set to 4.
- Now type in the command window: "mywave2=x" without the quotes. What happened? It turns out "x" is a special variable that sets the values in your wave to 0, 1, 2, 3, 4, 5, 6, etc.
- "y" is also a special variable. What does it do?
- Use the command window to set "mywave2" so that the values of the wave are multiples of 5. So it should be 0, 5, 10, 15, 20, etc.

- Duplicate “mywave2” into a new wave, “mywave3”, and add it to the table. Set each value in “mywave3” so that it is equal to the value of “mywave” plus the multiple of five in “mywave2”. You should be able to do it in one command. You should also be able to do it without calling “mywave2”, can you?

2. Load data into IGOR and graph it.

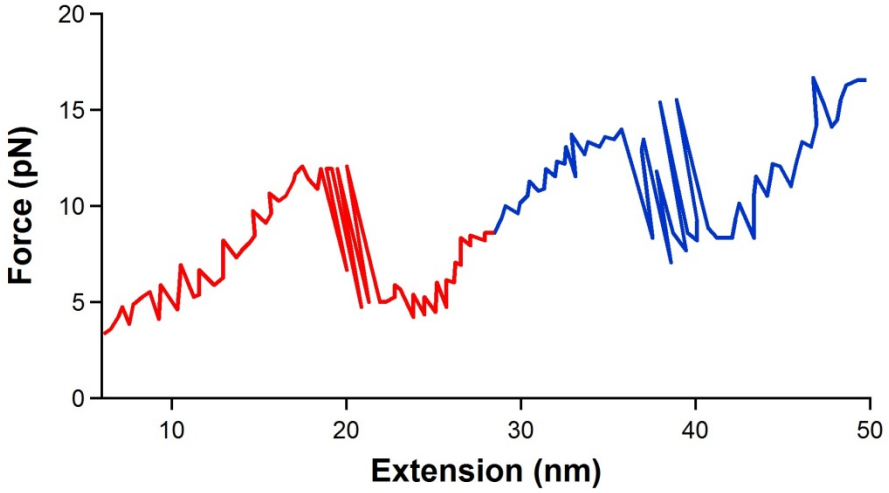
A) Watch the ezLecture: Loading Data in IGOR. Load both data files: Junker_Fig1B.txt and Junker_Fig1C.txt. You can load these both as general text files.

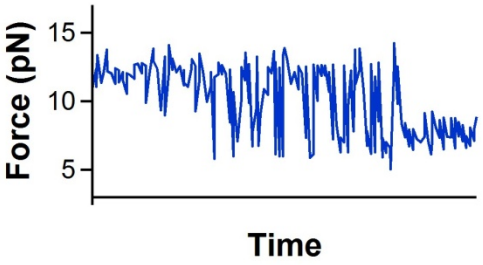
B) Now watch the ezLecture: Graphing with IGOR. This lecture will go rather quickly and that is okay. You can pause and rewatch if needed. The key points are the following:

- To make a new graph, you need to go to the “Windows” tab at the top and select “New Graph”.
- To change the look of the trace on the graph, double click the trace.
- To change the look of the axes, double click on the axes.
- To change the size of the graph, double click on the white margin outside of the graph.

You can also right click on the trace, axes, or margin to pull up the quick menu, or you can modify the graph by clicking on the graph and opening the “Graph” tab at the top.

C) Make two graphs that correspond to Figure 1B (bottom) and Figure 1C. The graphs should look exactly (size, colors, axes, fonts, etc.) like the ones below. Hint: Append two curves to Fig. 1B, so that one curve will be red and the other will be blue. When you append the curves click on the “More Choices” button to set the appropriate range.




Force (pN)
15
10
5
Time

3. Fit curves in IGOR.

A) Now we are going to want to fit the three curves that are shown in Fig. 1B in Junker et al. To do this, we are going to first select the region that we want to fit, then fit a curve, and finally repeat the process two more times until we have fit all three curves.

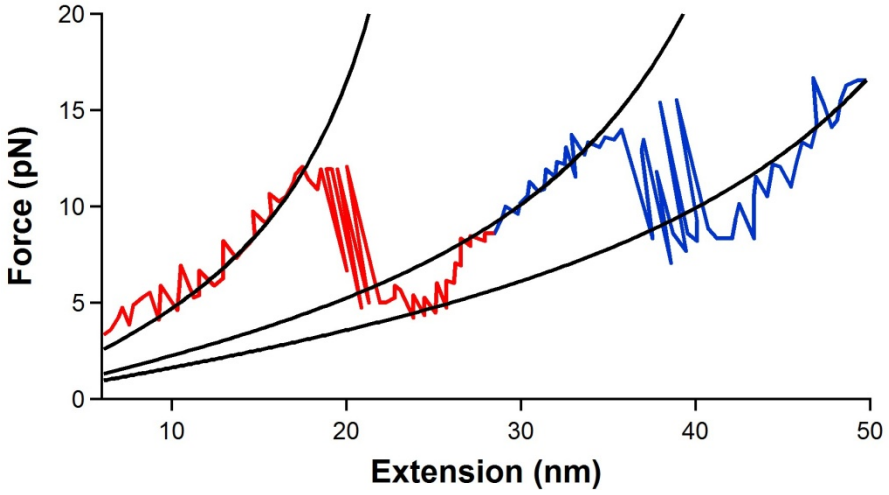
To select a particular region you are going to want to use the cursors. Add them to the graph by clicking on the graph, then clicking on the "Graph" tab at the top, and selecting "Show Info". Click on the circle or square and then drag them onto the graph. The circle should select the first point in the region, the square will select the last point in the region.

To fit a curve you will need to go to the "Analysis" tab and select "Curve Fitting". Under "Function and Data" select the appropriate x and y waves and the correct function. Here we want to fit a worm-like chain model, which does not happen to be one of the functions. You will have to enter this function by hand. This function is in reference 16 of the paper. To enter a function click on "New Fit Function" and enter the following equation:

(4.1/p) * ((1 / (4 * (1 – (x/L))^2)) - (0.25)+(x/L))

L and p should be entered as fit coefficients and x should be entered as the independent variable. Then select "Save fit function now" and select the function on the "Function and Data" tab. On the "Data Options" tab click "cursors" so that the function will only fit the selected region. In the coefficients tab you should put in your initial guess for the persistence length, p, and the length of the molecule, L. The persistence length is a polymer property and should be set to 0.5 nm for this data. Put in 0.5 and click on the "hold". The contour length is probably not more than 100 nm. Put in 100 and do not set the "hold". This is the variable that you are looking to fit. Then in "Output Options" click on "X range full width of graph" to get the fit curve to go across the whole page.

Fit each of the three regions and plot the fits on top of the graph. You'll have to rename the fit waves or it will overwrite them. You'll also notice that one of the fit waves will give some erroneous data outside of the range that you fit. You can append this fit wave using "More Choices" so that the erroneous region does not show up. You should get the following plot:



4. Make and save figures using IGOR

A) First let's save a graph as a .jpg file. Click on your graph for Fig. 1B, then go to the "File" tab, and select "Save Graphics". This will pull up a dialog box that will allow you to save the graph to a number of file formats including the jpg format. Save your graph and open up a Word document and insert the graph. It should look very nice.

B) But let's say that we want a figure that contains multiple graphs. To make a multi-graph figure you need to make a "layout" in IGOR. Go to the "Windows" tab and select "New Layout". Append your graph for Fig. 1B to the layout along with your graph for Fig. 1C. You should now have a layout with two graphs in it. To save the layout, click on the layout, and then go to the "File" tab, and select "Save Graphics". This will again pull up a dialog window that will allow you to save the layout.

C) However, before we save this layout, let's try to make the layout look like the figure in the paper.

- Zoom in by going to the "Layout" tab at the top, selecting "Zoom", and picking the level of zoom that you would like. If you don't see the "Layout" tab, click on the layout and it should appear.
- Embed both graphs by right clicking on them and selecting "Convert Graph to embedded". You will lose your graph window when you do this. To unembed the graph you can right click in the outer margin of the graph and select "Convert to Graph and Object". Move the graphs to where you want them and embed them.
- Remove the frame around the graph by right clicking in the outer margin of the graph and selecting "Frame" and then "None".
- On the left side of the layout there are some icons. If you mouse over them, they will tell you what they are. There are two sets of tools: Layout Tools and Drawing Tools. Click on Drawing Tools and you should get a whole new set of icons on the left. If you click on Layout tools you will get the original icons back.
- Let's explore the Drawing Tools. Go ahead and draw all of the following items so that your figure starts to look similar to the one in the paper. Items are: 1) arrows in Fig. 1B between the fit curves with text above the arrow telling the length, 2) scale bar for the time axis in Fig. 1C, and 3) dashed lines in Fig. 1C (these are actually fits, but I want you to get some practice drawing dashed lines).
- You can also add images to your layout. Go to the "Misc" tab and select "Pictures". This will open a dialog box where you will be able to select an image file using "Load New Picture from File". Go ahead and load the file Junker_1B_pics_at_top.jpg. Then select "Place Picture in Draw Layer" and click on the layout. You should see the image appear. You can place the image just above the data in Fig. 1B.

Now go ahead and save this layout as a jpg. This will be what you turn in!

5. Use procedures in IGOR

A) Watch the ezLecture: IGOR procedures if you know how to program. This video will not make much sense if you do not know how to program. However, if you are already familiar with programming, this will allow you to skip most of the tutorial. If you don't know how to program, watch it again after you finish with part E of this tutorial.

B) Procedures are ***programs*** that IGOR executes when you call them from the command line. To load and run a procedure, first open the procedure by going to the "File" tab, selecting "Open File" and then "Procedure" and browsing to the .ipf file that you would like to open. Once the procedure is open, then just click on the command window (which compiles the procedure), and type in the command. For example, open the procedure "MyFirstProgram.ipf" and type on the command line "MyFirstProgram()" without the quotes. You should see a message print out in the command line. Now load "WLC_Fit.ipf". This is a procedure that will load a fit function so that you don't have to type it into the Curve Fit dialog box. Click on the "Analysis" tab and select "Curve Fitting". You should now see the WLC_fit listed as one of your available choices. Don't you wish you would have known about this before completing section 3!

C) You can make a new procedure by going to the "Windows" tab and selecting "New" and "Procedure". Enter the name of your procedure as "Calculator". Then type the following:

```
#pragma rtGlobals=1                    // Use modern global access method.

function Calculator()

end
```

This will make a new program that doesn't do anything. Good programming etiquette dictates that the name of your ***function***, Calculator(), is the same as the name of the file: Calculator.ipf.

D) Let's go ahead and make the program do something. Let's make it calculate some values.

If you know how to program using a ***procedural*** or ***sequential*** programming language like java, C, C++, or python, then this will be child's play. Your job in this case is to make a calculator program that either subtracts, adds, or multiplies two numbers. The inputs are the two integers to be subtracted, added, or multiplied, and an additional input that tells the program which operation to select. The program then should print the answer. To make it a little more difficult the program is not allowed to use the * symbol for the multiplication step, instead use a for loop. Check your program against the one listed at the end of section E.

If you aren't a master programmer, then this will all seem very new to you, but I assure you that it will be very useful! To input something to the program you'll need to ***pass*** a parameter to the function by typing the name of the parameter in the parentheses. You can pick any old name you want except for those names that are ***reserved*** like: x, y, time, variable, etc. Once you have selected a name, then on the next line you have to tell the computer what that name represents. Does it represent a numerical

variable, a string of text, or a wave? In this case I wanted to pass a numeric variable to the function, so on the first line of the code I'll ***declare*** the parameter I passed to the function as a ***variable***. Finally, I'll print the answer of the addition of the two variables on the last line before the program ends.

```
#pragma rtGlobals=1                    // Use modern global access method.

function Calculator(var1, var2)

        variable var1, var2

        print (var1+var2)

end
```

Go ahead and enter this ***code*** into your program and press "compile". When you press "compile" the computer ***compiles*** the program, which means that it checks to see if there are any "grammatical" errors in your code. If there aren't any ***compilation errors***, you can run the program. To run the program type "Calculator(3,4)" without the quotes on the command line. Presto! You should see that the command line prints a 7. If an error popped up while you were running the program, then this is known as a ***runtime error***. Both types of errors are bad and require you to ***debug*** the program, or look for what line or lines of code are incorrect. Sometimes there will be no compilation errors and no runtime errors, but the program will still not work as intended. For example, what if you had accidently entered the minus sign into the program, instead of the plus sign? There would not be any compilation or runtime errors, but the program still wouldn't work as intended. If your program works correctly, then good work! You have created and executed your first program in IGOR!

But you are not done yet! A good programmer always ***comments*** their code so that others will know what is happening. To add a ***comment*** or a statement that won't be processed by the compiler or executed during run time, add two front slashes to the line with the comment. Everything after the two front slashes will be ignored. Typically, programmers will give a brief introduction to their code at the beginning of the program and then comment important lines. I have commented the program in the following way:

```
#pragma rtGlobals=1                    // Use modern global access method.

//This function is a calculator that can add two numeric variables.
//You execute the program by typing Calculator(var1, var2) on the command line.
//        inputs:  var1 and var2 are numbers
//                 outputs:  none
//                 displays:  the program prints the sum of var1 and var2

//Example:
//Calculator(3, 4)
//7
```

```
function Calculator(var1, var2)

        variable var1, var2                    //both numeric variables you want to sum

        print (var1+var2)           //display the sum

end
```

E) Now let's actually make the calculator into a calculator. To do this, we need learn about ***flow control***. In a typical program each line is executed in order and the flow is sequential. However, there are statements that will allow you to control which lines of code are executed and to also control when those lines are executed, changing the flow of the program. There are only two flow controls that you need to know about: the ***if-else statement*** and the ***for loop***. There are other forms of flow control such as the while statement and the switch, but you needn't worry about these here.

If we want to make a calculator that adds, subtracts, or multiplies our two numbers then we will need to control the flow of the program so that we don't add, when we want to multiply. To do this we will need to use the if-else statement. Let's pass another parameter to the program that is a string of text string that says either "add", "subtract", or "multiply". This time we will need the quotes when we pass the parameter so that the program knows it is a ***string***. We'll also need to declare this variable as a string right after we declare the first two numeric variables. Then we can use the if-else statement to ask if the string is equal to "add". If it is, we will add the numbers. If not, we will ask if the string is equal to "subtract". If the string is equal to subtract, we'll subtract the numbers. If not, we will multiply the numbers. Pay attention to the ***syntax*** for this if-else statement or you will get a bunch of errors. If you forget the syntax, you can always add the statement by clicking on "templates" at the bottom of the program, select "flow control" and the statement that you would like to use.

```
#pragma rtGlobals=1                    // Use modern global access method.

//This function is a calculator that can either add, subtract, or multiply two numeric variables.
//You execute the program by typing Calculator(var1, var2, selector) on the command line.
//        inputs:
//                                var1: a number
//                                var2: another number
//                                selector: a string that is equal to either "add", "subtract", or "multiply"
//                        outputs:  none
//                        displays:  the program prints the calculated value of var1 and var2
//Default setting is to multiply the two numbers.

//Example:
//Calculator(3, 4,"add")
//7

function Calculator(var1, var2, selector)
```

```
	variable var1, var2			//both numeric variables you want to sum
	string selector		//should be either "add", "subtract", or "multiply"

	if( stringmatch(selector,"add") )
		print (var1+var2)		//display the sum
	elseif( stringmatch(selector,"subtract") )
		print (var1-var2)		//display the subtracted value
	else
		print (var1*var2)		//DEFAULT is to display the product
	endif

end
```

There are a couple of other things you should notice. First, you should notice the indentations that I am using. By indenting your program correctly you and others can easily read the program and parse all of the control statements. Second, you should notice that I am using the command "stringmatch" to see if the string variable is equal to "add" or "subtract". There are a bunch of commands already programmed into IGOR that do all sorts of things. To browse these commands go to the "Help" tab and click on "command help". This will pull up a browser that will show you all of the commands. Look up stringmatch and see what it does. Finally, you should notice that I have set the default action to multiply the two numbers. One could actually input "yabba dabba" as the string and it would multiply the two numbers. If you would like your program to print out an error message if the string is not "add", "subtract", or "multiply" then you should add another elseif clause to the chain.

Now let's say that we could not use the multiply symbol in our calculator program. How could we make this program multiply something? One way is to write a line of code that adds the first number to a variable, and then to have the program repeat this line of code the number of times indicated by the second number. To repeat particular lines of the program over and over again, we need to use a form of flow control, and that is the for loop. The for loop is a statement that execute the lines of code in the ***body*** of the for loop the number of times indicated by the ***conditional statement***. Let's put the for loop in our program to see how it works. We'll also need to declare a couple of variables in the body of the program to keep track of the sum that we accumulate and the number of iterations of the loop that we have completed.

```
#pragma rtGlobals=1		// Use modern global access method.

//This function is a calculator that can either add, subtract, or multiply two numeric variables.
//You execute the program by typing Calculator(var1, var2, selector) on the command line.
//	inputs:
//			var1: a number
//			var2: another number, if you want to multiply this must be an integer
//			selector: a string that is equal to either "add", "subtract", or "multiply"
//		outputs:  none
//		displays:  the program prints the calculated value of var1 and var2
//Default setting is to multiply the two numbers.
```

```
//Example:
//Calculator(3, 4,"add")
//7


function Calculator(var1, var2, selector)


        variable var1, var2                     //both numeric variables you want to sum
        string selector             //should be either "add", "subtract", or "multiply"


        if(stringmatch(selector,"add"))
                print (var1+var2)               //display the sum
        elseif(stringmatch(selector,"subtract"))
                print (var1-var2)               //display the subtracted value
        else
                variable i              //variable to iterate the loop with
                variable product  //variable that keeps track of the sum and will eventually be the product
                product=0               //initialize the product to zero

                for(i=0;i<var2;i=i+1)           //initialize the iteration variable to zero;
                                                //if i is less then var2 then execute the body;
                                                //once the body has been executed increment i by 1
                        product=product+var1      //body of the for loop---which is to add var1 to the sum
                endfor

                print (product)                 //DEFAULT is to display the product
        endif

end
```

Of course the program that we have made is limited in the fact that var2 must be an integer for the multiplication portion of the calculator to work. We will note this as a ***feature*** of the program. If the program needs to work with fractional values of var2, then this would be a ***bug*** and would need to be fixed. Many times the difference between a feature and a bug is just a few comments to the user!

Look at the amazing program that you have built and hopefully understand! You may be thinking to yourself, "This is easy. Why did I wait so long to learn this?" Or you may be thinking to yourself that this still seems like a foreign language. If the latter is the case, then you may want to take some more formal lessons on programming using online educational services. There are some great free online courses that I could recommend if you are interested.

F) Now that you have been introduced to programming let's try to estimate k_f and k_u at ~10 pN of force. Because we want to focus on programming and not on science, we'll try to do this using a "simple"

method. Specifically, we'll divide the trace from Fig. 1C into either folded or unfolded by using a numeric threshold of 9.3 pN. Data above 9.3 pN will be considered folded and data below 9.3 pN will be unfolded. Now use a for loop and iterate through the data. As soon as the trace crosses from folded to unfolded or vice versa note the time. You'll need to use some if-else statements to get it to work. Then make a wave with the times spent folded and another wave with all of the times spent unfolded. For example, if the molecule spent 2 s folded, then 0.1 s unfolded, then 3 s folded, and finally 0.2 s unfolded, then 2 and 3 would be in the wave with the times spent folded and 0.1 and 0.2 would be in the wave with the times spent unfolded. To find k_f, take the inverse of the average time spent unfolded. To find k_u, take the inverse of the average time spent folded. Now go ahead and write the program, what did you get for k_f and k_u?

G) If you made it this far, you should be a whiz at IGOR. Congrats, because the rest of the data analysis this semester will be a lot easier!