Introduction to *Igor Pro*, Spring Term - 2024

Carl Grossman, Swarthmore College

Gabe Spalding, Illinois Wesleyan University

There are many data analysis packages available on every computer platform (Windows, Mac, Unix, *etc.*). One of the more common ones found in real professional and graduate research labs is called *Igor*, made by Wavemetrics. For outstanding control and precision of the graphics, the earliest versions of *Igor* employed powerful command-line instructions. *Igor* also provided a flexible environment, making it easy to create, and even to automate, quite complex analysis and visualization procedures. In its current incarnation, *Igor Pro*, the software has become ever more user friendly, yet the capability for sophisticated data analysis and visualization remains. The program can be used for simple tasks like displaying introductory lab data in a few straightforward steps and yet, at the same time, you can move on seamlessly as you encounter more and more complex experimental apparatus (see the Wavemetrics website for screenshots).

1 Simple Data Loading

Open *Igor* and select from the menu Data, the sub-menu Load Waves, and then from the drop-down list Load *Excel* file.... Next, below the Path box, find the File button and press it to browse to the file of interest (I've supplied an example file called SimpleData.xlsx that you can use here once you've downloaded it to your computer). Open SimpleData.xlsx, but don't click "Do It" yet: Igor very kindly offers you the opportunity to tweak a few details in the loading process. In the dialog box now open, under Output Waves, select the option to take the wave names from worksheet row 1 (which makes sense given the way data was entered into SimpleData.xlsx, which contains data relevant to a recent court case for which a lawyer requested an expert opinion on the basic physics involved). In any case, make sure you've checked the checkbox just a bit further down that says, "Make a Table." Once you're happy with the options you've selected, click "Do It," and a table with your data will appear in Igor.

2 Simple Graph

Go to the *Igor* menu called Window and select New Graph. Select whatever variables you wish for the x- and y-axes of the graph (*e.g.*, if you are using the data loaded in the previous section, select Speed as the measurement to be associated with the y-axis and Elapsed Time as the independent variable associated with the x-axis), and hit "Do It."

3 Modifying the Look of a Graph

Resize the graph, by dragging one corner. Then, double-click on the data trace within your graph. In the subsequent dialog box, change "Lines between points" to "Markers" and explore the types of plot symbols you can use for each datapoint (choose a closed circle for now.) Also click on "Color" and choose a nice purple. Note that you can also change the size of the plot symbol, if you wish (or type "Auto" if you want to go back to *Igor*'s choice). Click "Do It" to update your graph. Since you will often opt for the formatting you've implemented, next go to *Igor*'s Graph menu and select "Capture Graph Prefs." In the dialog box, select aspects you would like repeated for subsequent graphs, such as select "Window Position and Size" and "XY Plots: Wave Styles (lines, markers, colors, ...), and then click Capture Prefs."

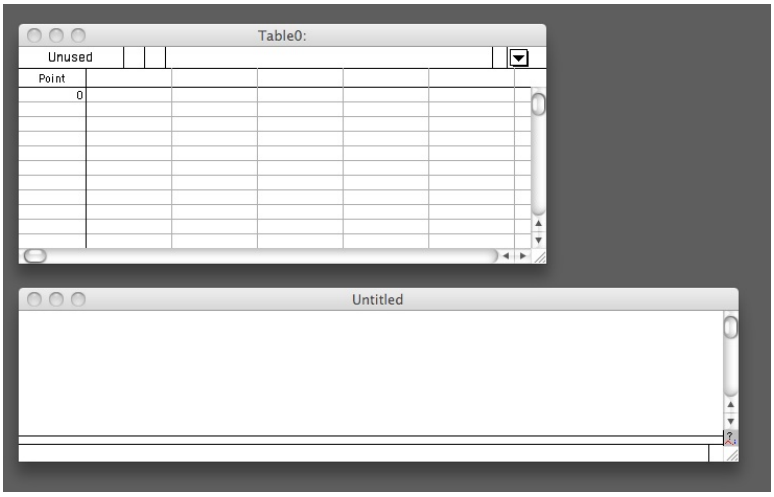
Igor will remember.

To make other modifications to the graph, double-click on one of the axes. In the dialog box, find the tab called Axis Label: choose a font and font size. Type a label, such as "Elapsed Time (sec)" for the bottom axis. To make the label boldface, place your cursor in front of whatever text you wish to modify and select the "Special" button at left, choose Style and click Bold and then Okay. *Igor* gives you plenty of other options: part of what you type can be in Symbol font or, using the "Special" button at left, you can make part of what you type a superscript or subscript and then revert to Normal text. It might be nice to increase the font size. If you've done anything that takes time, once you've made choices (of font, size, or style) that you want to apply to the other axis, you can copy the code that *Igor* has generated, then click in the upper left to select the other axis, and paste the code into the label box. Modify that pasted code to yield an appropriate label, such as "Speed (m/s)" for the left axis. Once you're happy with changes you've made, click "Do It" to accept them.

The following illustrates just *some* of your options. Double-click one of the axes. This time, instead of clicking the tab called Axis Label, try the first one on the left, called Axis. Uncheck "Axis standoff" and turn on "Mirror axis," then repeat for the other axis (using the button at upper left to switch between axes). In Ticks and Grids, select the Location as Inside, and set the Major Tick Length to 4. Again, do the same for each axis. For the left axis, select Auto Man Ticks, turn off Auto Ticks in favor of Computed Manual Ticks, with "Tick Increment" = 5 and, under Minor Ticks, with "Number per major tick" = 1. For the bottom axis, try "Tick Increment" = 4 and, under Minor Ticks, with "Number per major tick" = 3. You can choose other numbers if those don't look good to you. In Axis Range, select a Manual Range from 0 to 17 for the bottom axis and from 0 to 45 for the left axis. Once you are happy with your choices, click Do It to retain those changes for this particular graph.

4 Command Line Basics

When you first start *Igor*, it looks a bit like *Excel*. Yet *Igor* is so much better than *Excel*. *Excel* is awkward in many ways (*e.g.*, operations are, by default, on individual cells, rather than entire data sets), and quite limiting in others, which can become important as you progress! In fact, there are many reasons to say that *Excel* is *not* really a modern spreadsheet program. The small amount of effort that is required for you to learn *Igor* is a very good investment, which will certainly pay off throughout your time at Illinois Wesleyan (and perhaps well beyond)!

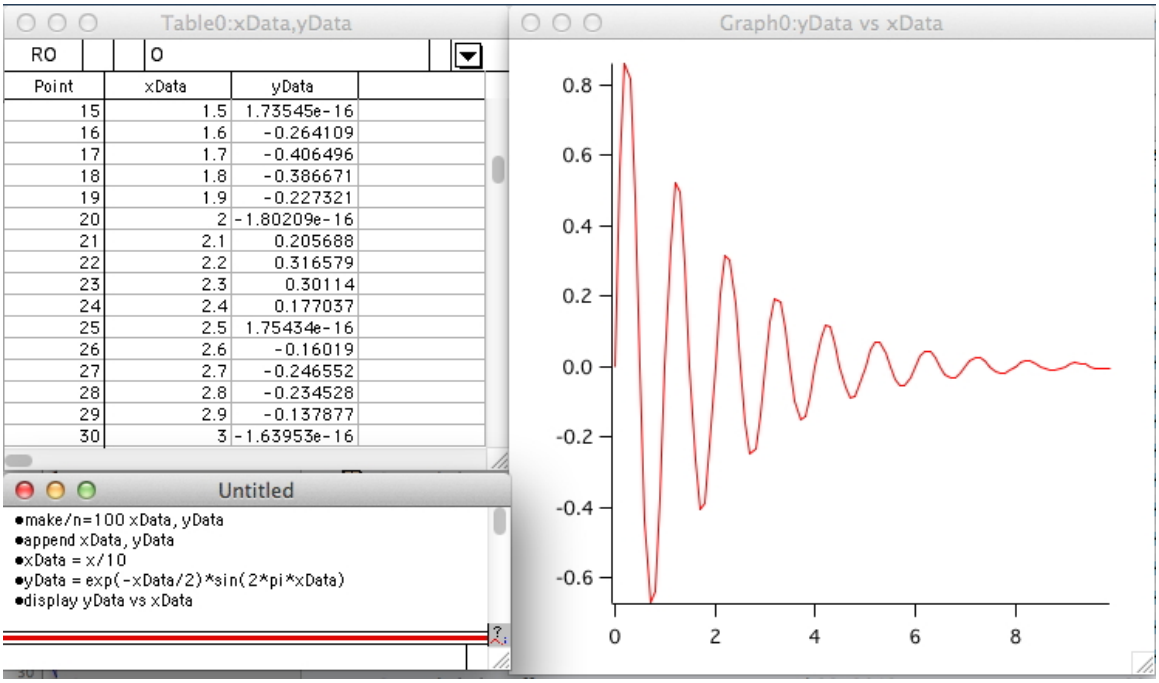


As shown by the screenshot above, *Igor*'s default opening reveals an empty Table Window called "Table0" (as well as a Command Window, which initially carries the self-denying title, "Untitled"). This Command Window has two parts, a command entry bar at the very bottom and, importantly, a history box to display previous operations (which turns out to be both a real time saver later on, as you will see, as well as an *archival record* of everything you've done in analyzing your data). Once you've spent just a little time using *Igor*, some of you will find that you would rather type commands than point-and-click a mouse; it turns out that you do, indeed, have the flexibility to do things either way. Regardless, the history display records it all, not just typed-in commands; your mouse clicks and dialog box selections also result in entries. Again, the history display will prove very handy for reviewing a series of steps in data analysis or for calling up and executing a previously typed command (you can use the up-arrow and down-arrow keys to navigate through the history and hit enter to copy any part you wish into the command bar for editing and re-use). Moreover, when you save an *Igor* "Experiment" (as it is called), the entire history is saved along with your data, formatted graphs and tables, procedures, functions, programs, *et cetera*. This makes life a great deal easier when you later wish to return to your work.

Igor will remember.

If you've just worked through the previous sections of this tutorial, the history portion of the Command Window is already populated. To repeat a command, just highlight it and hit return twice. (The first time you hit return, that previous command is entered into the active command line, where you could edit it, if you wish. The second time you hit return, it runs the command.) Of course, you can also enter *new* commands into the command line. In fact, it is easy to do quite a lot, rather directly, with *Igor*'s commands, as you quickly infer if you select *Igor*'s Help menu and click on Command Help.

For now, as a first example, you'll just learn the syntax used to create x and y data arrays; you will also calculate an arbitrarily complex function, such as $y = e^{-x/10}\sin(2\pi x)$ for the range $0 \leq x \leq 10$, and plot it.



By exploring the command structure used by *Igor*, you are also learning how to write programs in *Igor*, which can allow you to *completely automate* many data acquisition and analysis processes. (Given proper connection, *Igor* can even control external test and measurement instrumentation, image capture and analysis, automate identification of features in data, *etc.*)

Igor is at your command.

A first exercise in giving *Igor* your commands:

The previous figure showed a sequence of commands, along with their results, which we now review. Open *Igor*, click on a table to ensure that it is the center of *Igor*'s attention, then click in the command line, and enter those commands, which are repeated below. (As you do so, watch what happens when you hit "Enter" at the end of each line, and see whether you can figure out what the "x" corresponds to on the right side of the equation in the third line below):

```
make /n=100 xdata, ydata
append xdata, ydata
xdata = x/10
ydata = exp(-xdata/2)*sin(2*pi*xdata)
display ydata vs xdata
```

In the first line, you made new "waves" (*i.e.*, variables stored in memory. The /n=100 tells *Igor* that these waves will contain one hundred data points. The variable names for each of the waves created (here "xdata" and "ydata") are separated by commas.

In the space below, please write down what the "x" corresponds to, on the right side of the equation in the third line above:

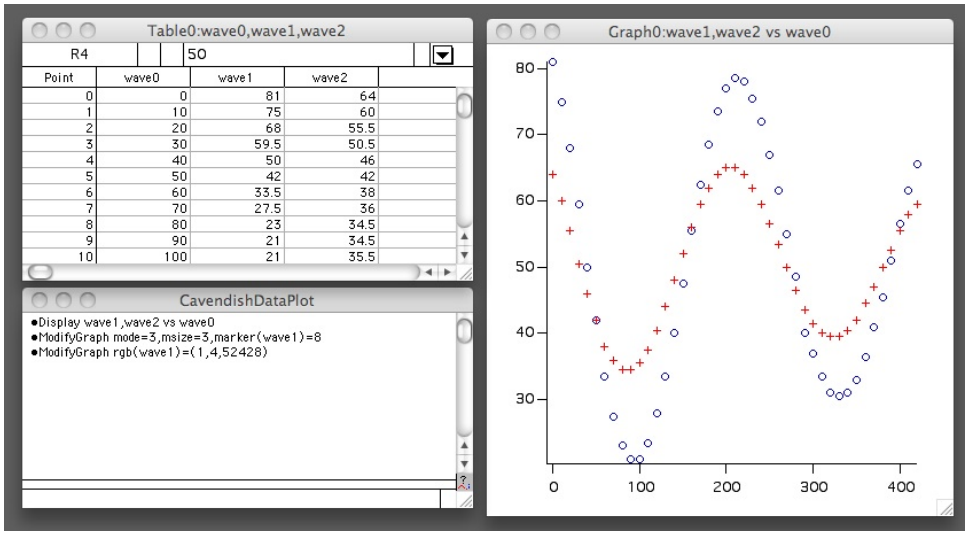
The "append" command operates on the "top-most" window, in this case Table0; it adds elements to the table. If a graph window is selected, the append command adds plots to the graph. One of the menus that appear at the top of the screen will also change depending on which window is selected. In other words, tables, graphs, and layouts automatically have different options available. Try clicking on the Graph and Table windows to see which menu changes. Select the Graph window and try the append command.

Note that the "ydata = exp ..." in the fourth command line above, calculates the entire data array in one command, unlike *Excel* (which typically forces you to specify a range of cells). With your Graph window visible, go back to the Command Window and try editing the function you previously entered, perhaps multiplying it by *10 (click on the command in the history display and hit enter and the command will be automatically typed into the command entry box - there, you can edit it and hit enter again to execute the new command). ...Are you staring to like *Igor Pro* yet?

5 Table and Graphing Basics

Again, as an alternative to typing instructions into the Command Window, *Igor Pro* also has an intuitive graphical user interface for basic data entry and graphing. The Table Window can be used either to enter data manually, to simply cut and paste data from other programs, or to display data that *Igor* (very capably) can read from a file or create from a function. Once data is entered, a new graph can be made either by using the Command Window (as you did above) or just by selecting the menu item Windows→New Graph. In fact, some people never bother using the Command Window at all: everything you need for introductory-level work (at least) is available via the menus associated with the graphical user interface.

The next data set we'll take a look at was taken during a "Cavendish Experiment" where the data was collected and entered by hand, then displayed, and later fit to a (non-trivial) function that describes a damped harmonic oscillator. As data is typed or pasted into *Igor*, the default names are given, wave0, wave1, wave2, etc. If preferred, these names can be changed with the Rename operation; that's a very useful trick for keeping track of the physical meaning of each column. (Using the waves created in the previous example, you'd just type "Rename ydata fred; Rename xdata wilma" in the Command Window: note that all tables and graph titles immediately update!) In our next example, wave0 is the time in seconds, wave1 and wave2 are the measured positions of a laser beam on a wall. (In this particular "Cavendish" experiment, the laser beam was reflected off a mirror attached to a torsional oscillator and so the laser beam position was related to degree of rotation of the oscillator. Assigning physically meaningful names won't matter for now, though, since we're just trying to demonstrate how to start using *Igor Pro*.)

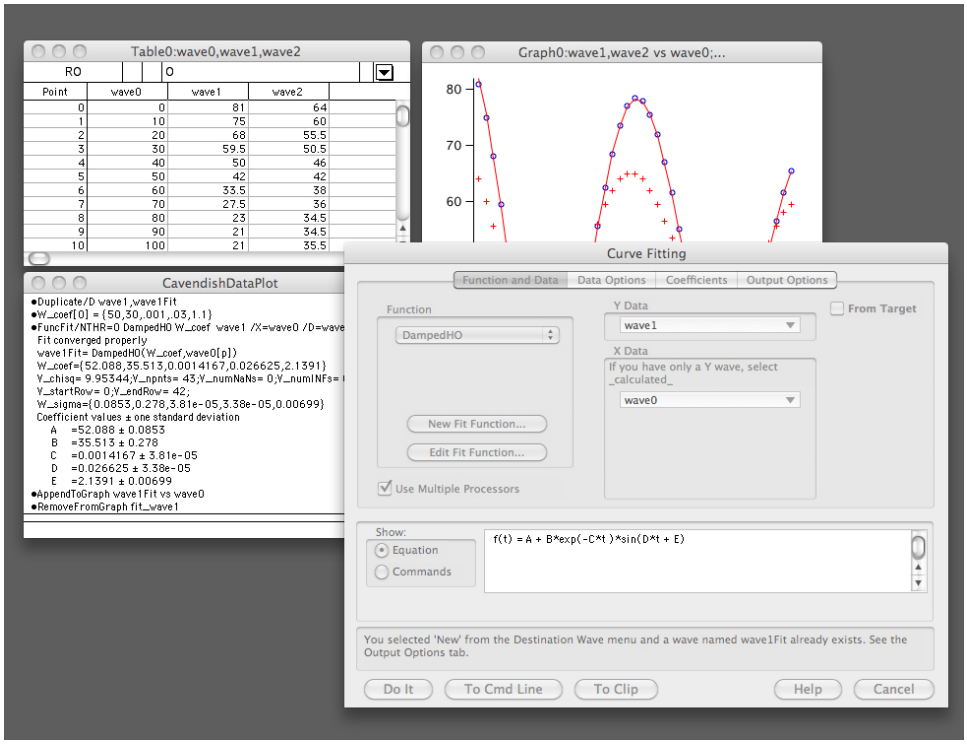


Data Plot Exercise:

- Open, in a text editor, "Cavendish.txt" from the tutorial files I've supplied for you. Select and copy the first bunch of data, running from the first entry, 00.00, through to the entry just before a big gap, 420.00 (In the actual experiment, these were the times at which the laser beam positions were noted).
- In *Igor*, start a new experiment. (You needn't save the previous example.)
- From the Window menu, select New Table and dismiss the resulting dialog box simply by hitting return (or enter). Then, in the Table0 window, click on the top row of the first column and paste. The column title should become "wave0".
- Repeat, copying the next two bunches of data into the next two columns of *Igor Pro*. (In the experiment, these were the observed laser beam positions, recorded for two different configurations of lead balls used in the apparatus.)
- To demonstrate *menu*-driven graphing, choose the Windows→New Graph menu item. A box will show up with two lists, a Y Wave list and an X Wave list. For the Y Wave, choose "wave1" and shift-click on "wave2" to select both data sets for plotting. If you were to leave "_calculated_" as the X Wave then the x-axis of your graph would simply use the data point number 0, 1, 2, 3, and so on; however, for this experiment, we want to see the dependence on time (which is wave0), so go ahead and select "wave0" for the X-Wave.
- Click on "Do It," and double-click on the curve to change the plot mode to "Markers."

6 Curve Fitting Basics

[Background: the Cavendish lab measures a shift in position of a torsion oscillator due to the (weak) gravitational interaction between 1-kg lead balls integrated into the apparatus. Although the effect is tiny, the reflected laser beam travels a fair distance to the lab wall, so even a small angular shift produces measurable deflection of the laser spot. Such an "optical lever" is often used to turn small displacements into something easily measurable. In the data at hand, the gravitational interaction between the lead balls produces a change in position of the laser beam of about 1 cm. In working with a Cavendish balance, our interest is in measuring the shift in equilibrium positions as we change the distance between the lead balls. To improve accuracy, what we actually do is to measure *displacements* from the "center" position for damped harmonic motion in the torsional oscillator; those displacements are large compared to the uncertainty of each position measurement. It is only by using *curve fitting* on this collection of data that we will extract the desired precision regarding the center positions.]



The figure shows the first pane of the Curve Fitting dialog box that will appear in the exercise below. Here, you will specify the Y and X arrays and the fitting function.

Curve Fitting Exercise:

- Select the graph window and then choose the Analysis→Curve Fitting menu item.
- The Curve Fitting dialog box opens. Select the leftmost tab near the top of the dialog box, labeled "Function and Data."
- Choose "wave1" for Y Data and "wave0" for X Data.
- Browse the available functions and note that, for any you select, the definition of that function appears in the large equation box. (In general, you'll write that equation down in your lab notebook!) For our test data, we will fit to a damped harmonic oscillator function, which is *not* on the list of built-in functions (nor would it be for *Excel*); such situations are common in science, and require you to define for *Igor* a new fit function, customized to the experiment at hand. Click on the button labeled "New Fit Function."

- Make up a name for this function, set up five adjustable "Fit Coefficients" (by entering A, B, C, D, E, each in its own box), name an Independent Variable (t), and type in the following "Fit Expression:" A + B*exp(-C*t)*sin(D*t + E). Finally, click on the button labeled "Save Fit Function Now."

- Click on the "Coefficients" tab atop the dialog box. It turns out that, regardless of the software you use (whether it is *Excel*, *Igor*, *LabVIEW*, or *MatLab*),"generalized" curve fitting requires you to make initial guesses for coefficients, so that the algorithms used won't be so likely to get "lost" while trying to optimize the fit to your data. This turns out to be a wonderful exercise that you should engage in frequently: just by looking at your data plot you can usually make reasonable estimates for each of the coefficients. The "Graph Now" button allows you to see what your model would look like with those guesses. In this way, you can play around with the coefficients until you get in the right "ballpark." For now, we'll just supply you with those "adjusted starting guesses" that we came up with: 52, 30, 0.001, 0.027, 2.1. The next step is to allow Igor to adjust these values to optimize agreement with the data.

- Before clicking "Do It," click on the "Output Options" tab. This dialog allows one to control the whereabouts of the calculated function that best fits the data.

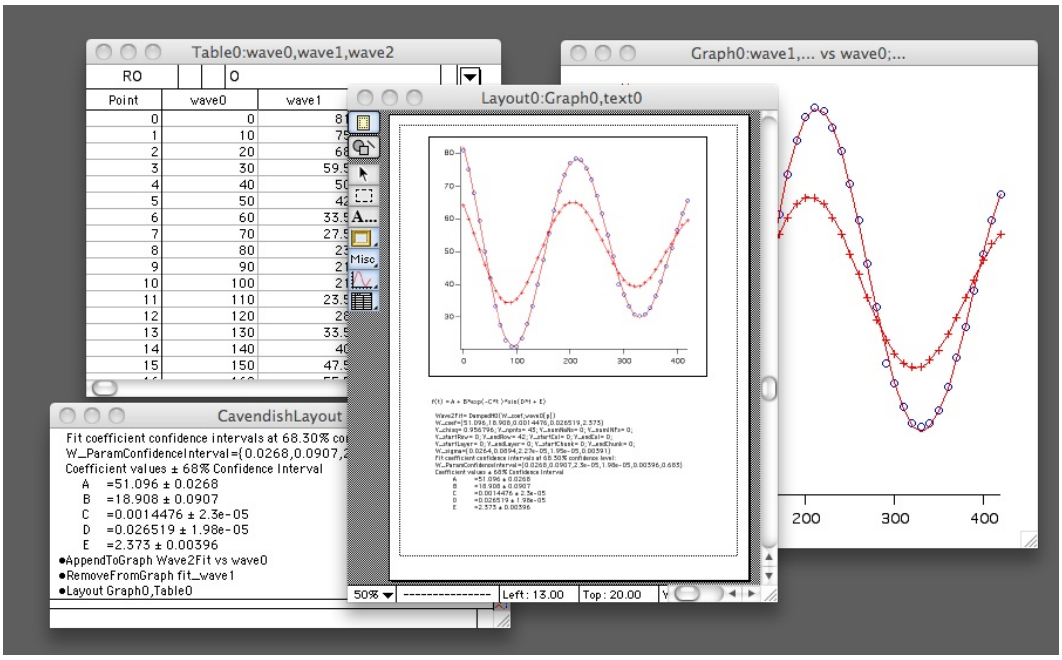
- Choose Destination→ "_New Wave_" and call it something reasonable, such as "wave1Fit."

- Finally, always double check settings on all of the available tabs and then click "Do It" (or just hit enter, which has the same effect). After that, hit enter again (or click "Ok") to dismiss the Curve Fit Window that pops up.

Parameter values giving the best fit appear in the Command Window, though the fit function is not automatically displayed (unless you use the auto feature in the "Output Options"). It is easy to add another trace to a graph: select the Graph→Append Traces menu item, and choose "wave1Fit" as a Y Wave and "wave0" for the X Wave. Alternatively, you could have simply used the Command Window, typing "append wave1Fit vs wave0." (If you did not previously change the graph mode to "Markers," you may now wish to double-click on the data that is plotted, so that you can change the way that different lines and points are displayed; this helps you to distinguish the data from the fit.)

We won't pontificate about the care one must take in curve fitting, but the use of some other software that makes it all too easy for students to develop some really bad habits; *hiding away* all of the controls might be part of the problem with those other packages. *Igor Pro* gives you a bit more access to what's going on "under the hood." Moreover, the curve fitting package in *Igor Pro* really is professional and robust. It won't take long at all to demonstrate, and then, very quickly, you'll be doing great things! – For now, though, simply save the experiment with your name in the title and close it.

7 Layout Basics



Layouts are used to adjust the size and placement of plots (and tables) so that they fit nicely into lab notebooks or, more generally, just to make neat and informative printouts. Too many times students have turned in curve fit data with no information about what fitting function was used! Such "annotations" should be entered directly onto the Layout (instead of adding them directly onto the original graph): with this approach, you will have flexibility in re-sizing and repositioning each element, independently. I recommend using Layout annotations not just for showing the fitting function equation but also for the *results* of the curve fit (both the optimized values of fit coefficients and the uncertainty in each of those coefficients). So, if any of those annotations are currently in your graph window, double-click on them, then delete it from the original graph. You can use *Igor*'s history to paste that information into a Layout, as shown in the next exercise.

Layout Exercise: Open the file that was saved in the last exercise (double-click on the file or choose the File→Open menu item). Notice that all of the commands that you had previously executed are automatically done over. The history of commands regenerates the graphs, curve-fits, and modifications. *Igor* keeps great records of everything you've done!

- Choose the Window→New Layout menu item.
- Select "Graph0" and click "Do it" to open the Layout0 window.

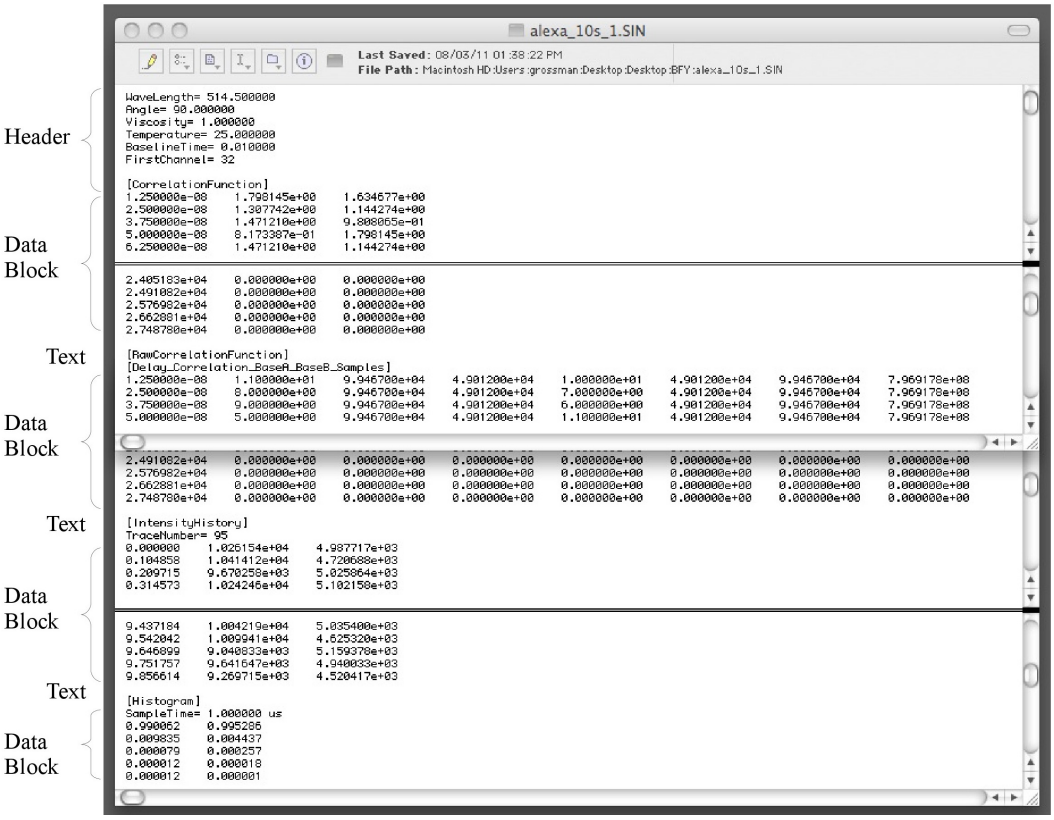
- Re-size and position your graph so that it only takes up as much of a standard page as seems appropriate.
- Copy, from the history window, the results of your curve fit, beginning with the line that reads "Coefficient values ± one standard deviation" and including the optimized values of fit coefficients and the uncertainty in each of those coefficients.
- Now return to the Layout window and notice the toolbar on the left side of the window. To add text to the layout, click on the A... button and click somewhere in the layout area.
- A dialog box opens with many options. Here you should now paste the text that you copied from the history window, showing the results of the curve fit, and should top this off with the explicit equation for the fitting function that you used:

$$y = A + B*\exp(-C*t\ )*\sin(D*t + E).$$

- Before you hit "Do It," note that there are lots of bells and whistles available in this dialog box: font (including the Symbol font), subscript and superscript, size and position control are all pretty useful.
- After hitting "Do It," be sure to un-select the A... button by clicking on the arrow button, otherwise you keep making text boxes. Once you've selected the arrow button, you can *independently* re-size or position the graph and the annotation(s) you've made. Save your work, and any Layouts you've created are also saved!

8 Data Import Basics

More than with any other software I've tried, it's been relatively easy using *Igor Pro* to import data that is embedded in text and headers. *Igor* is also *especially* useful if you have *many* data files with the same structure (for example, files where the x-y data values always start on line 17 of each data file). Once you figure out how *Igor* can read a file, the process can be *automated* in a script to read a whole folder full of files! While the previous sections of this tutorial should be completed by *all* students in their first semester of physics, the remaining sections are great for anyone wanting some extra power. Over your college career (and beyond), you are likely to gather data from all sorts of commercial instruments. It doesn't take long to realize that there are some pretty crazy data formats out there. To get started, the following exercises use two different methods to extract data from a file that has lots of text surrounding the numbers we want, and four different types of data blocks.

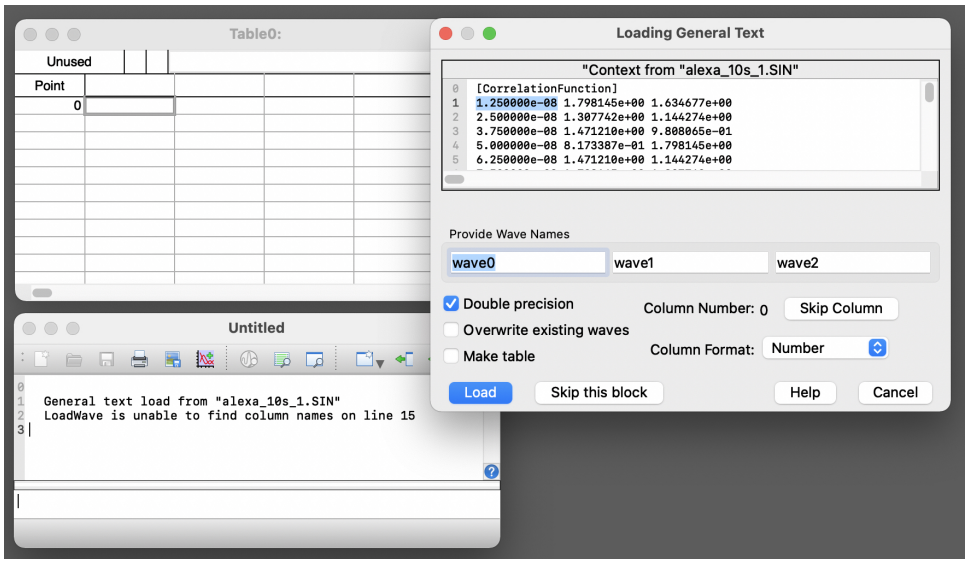


- The first method uses a dialog box to identify the data from within the file.
- The second method again starts with a dialog box to perform the same operation, but goes on to save it as a command. This command can be copied into a script, called a macro, to open any file of the same format and similar file name.

Here we have some test data that looks at *correlations* of a particular sort. The data file shown above starts with some header information, followed by a block of 3-column data, then a few lines of text, a block of 8-column data, text, a block of 3-column data, text, and finally two more columns of data. (A printout of these raw data files would span eighty pages!) Buried within all this, the correlation data that we want to extract for this exercise is contained within the first block, and the main point is that *Igor* allows you to excise that information with surgical accuracy. Within this block of interest, the first column is a delay time, the second column is the correlation function between detector 1 and detector 2, and the next column is vise versa (*i.e.*, the correlation you find if you start with detector 2 and compare to detector 1 after the given delay time). ...Did we mention that you're (soon!) likely to encounter some crazy data formats?

Data Import Exercise: Method I

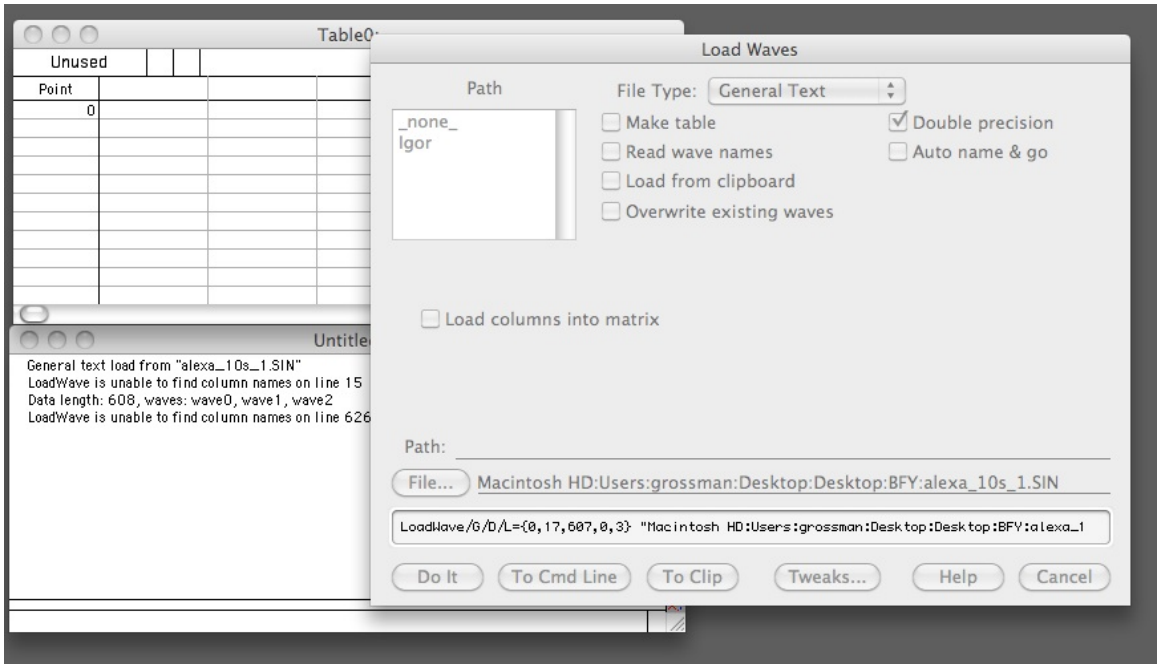
- Under the "File" menu, open a New Experiment.
- Choose the Data→Load Waves→Load General Text menu item.
- Select the data file "alexa_10s_1.SIN" from the tutorial files I've supplied. Select "Enable All Files." (.SIN is not a format that *Igor* natively recognizes.)



- Notice that even though this isn't an *Igor* file, *Igor* is smart enough to immediately skip the non-numerical text, and recognizes that there are three columns in the first block of data.
- These are the data that we want, but don't load them for the moment. See what happens when you click on the "Skip this block" button.
- Since that quick illustration caused us to missed the data that I'd like for us to focus on, go ahead and cancel the import and start again, this time actually click to Load the first set of data (three columns of two-detector correlation data).
- Now you're ready to quit the import, as we don't want to load the other data blocks.
- Notice that in the history there is information recorded about which line the data was found on: "LoadWave is unable to find column names on line 15," means that the data starts on line 16. Also, the history notes that there are 608 lines of data in the first block, as you should now check by listing the data in Table0. (Either use the "append" command or the Table menu.) Keep this experiment open as we continue.

Data Import Exercise: Method II

- Choose the Data→Load Waves→Load Waves menu item (yes, these menu items have the same name!)
- Choose General Text using the pull-down menu near the top of the Load Waves dialog box.
- The Double Precision checkbox should be checked.
- Click on File... button and once again select the data file "Alexa_10s_1.SIN" from the sample files I've supplied.
- This time click on "To Cmd Line." The import operation is not executed yet. The command line equivalent is entered into the command line. We can edit it and then execute the command by hitting return. But first let's make some changes.



- Look at what's now in the command line. To see what all of the slash-letter specifications (command flags) do, go to the Help→*Igor* Help Browser menu item and click on the "Command Help" tab near the top of the dialog box. You'll see an alphabetical list of commands on the left, scroll (or begin typing into the list) to move down to the "LoadWave" command.

- In some older versions, *Igor* would automatically add information about what lines the first data block begins and ends on, but the result would sometimes be off by one line, as shown in the figure above. In the command line, you want to edit the command immediately (*i.e.*, leaving no space) after the /D to read /L={0,16,608,0,3} and this should be followed by a space before the string (path) telling *Igor* where to find the file of interest.

- **Copy and paste that *path* information to a safe place, so that you can use it in the next exercise!!**

- If you were to execute the LoadWave command, *Igor* would open a dialog box asking for names for the waves. We don't want this if we are going to automate the process. So, instead, add the "/A" flag to the "LoadWave" command. (It shouldn't matter which order the flags are put into the command.)

- Hit return to execute the command. The history should show that data was loaded into wave3, wave4, and wave5.

- You can now display the correlation data in a graph of wave1 or wave 2 vs wave0. It doesn't look like much unless you change the plot to a semi-log plot. To do that, simply choose the Graph→Modify Axis menu item, select "bottom" from the pull down menu, and change the mode of this axis to "log."

- Again, the point is that even though this data was generated from instrumentation that had nothing to do with *Igor Pro* and was embedded inside a fairly complicated mix of text and header structures, it was still relatively straightforward for *Igor* to import the data. (Moreover, you'll soon see how to automate such processes!)

9 Creating Macros and Functions

Igor uses the Procedure Window for the development of macros as well as functions and full-fledged programs. Under *Igor*'s Window menu, select New, then Procedure. This window can also be opened from the Windows→Procedures→Procedure menu item (or cmd-M on the Mac). *Whenever* you write code, we strongly urge you to drag out the window to fill a large space, and click on the zoom icon at the bottom left, to enlarge the type by a factor of two (at least). This will make it easier for you to find your mistakes!

Type in a name for your procedure, preferably one that gives some hint as to its purpose, though for this preliminary example just type MyProcedure. In the subsequent window that opens, add a line beginning with either the word *macro* or the word *function*, followed by the name you wish to give to your macro or function. Let's suppose we would like the macro to print "Hello, World!" The full macro would read as follows:

```
macro MyMacro()
    print "Hello, World!"
end
```

To compile this (terribly basic) procedure, either click Compile (at the bottom of the Procedure Window) or click in any other window. To actually run the procedure, click on the command line and there type MyMacro() and hit return.

As a slightly more interesting example, suppose we wish to write a function of two variables. Below, we begin a new procedure with a line that starts with the word *function*, but now we also need to give names to the arguments of the function, separated by a comma and enclosed in parentheses following the function name:

```
function MyProgram(xwave, ywave)
    // initialize variables:
    wave xwave, ywave
    variable npts=numpnts(xwave)
    make /o/n=(npts - 1) deriv_xwave, deriv_ywave
```

To declare those arguments, we simply wrote the word wave and then the names of the arguments, separated by a comma. To declare simple variables, we wrote the word variable and then the variable names. To make new waves, we wrote the word "make." Details are available under Help → Command Help.

For flow control within our procedure, we will add a FOR loop with the syntax shown below, which will cause *Igor* to cycle through the following lines repeatedly, each time incrementing the index i by one. Our goal is to approximate the derivative by the average slope (the rise over run), and in anticipation of *graphing* the derivative, we associate each slope we calculate with the average x-value of the interval used. Note that square brackets are used to denote elements in a wave:

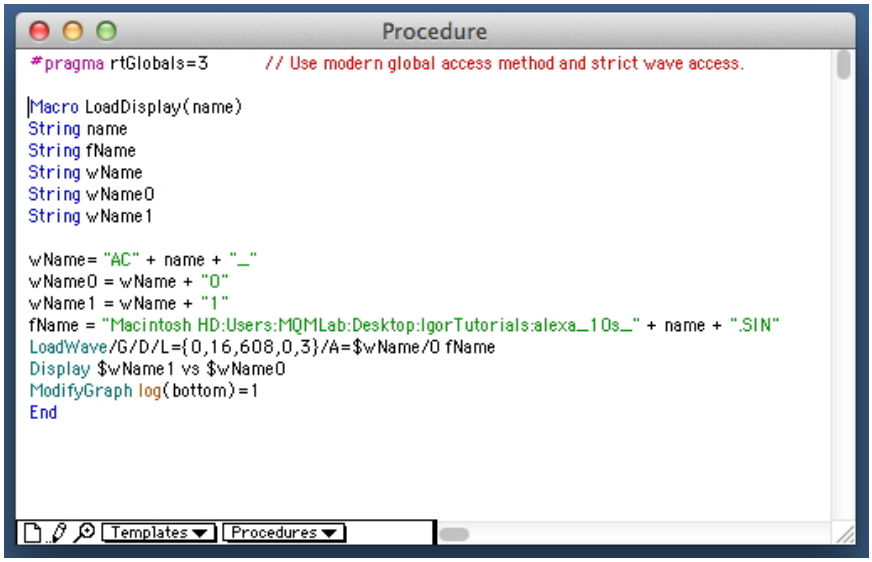
```
    variable i
    for(i=0; i<(npts-1); i+=1)
        deriv_ywave[i] = (ywave[i+1] - ywave[i])/(xwave[i+1] - xwave[i])
        deriv_xwave[i] = (xwave[i+1] + xwave[i])/2
    endfor
end
```

Again, to compile a procedure, either click Compile (at the bottom of the Procedure Window) or click in any other window. To actually run the procedure, click on the command line and type the name of the procedure, here MyProgram(). However, before you hit return you'll need to enter, as arguments of this function, wave names that already contain data; *e.g.*, if you have loaded my *Excel* file SimpleData.xls, then you would type MyProgram(ElapsedTime, Speed). After you do hit return, the waves your program has created are stored in memory, but you can append them to any table or graph you have begun, or can create a new graph or table to display them.

9.1 Intermediate Exercise A: a macro to read in and display data

Suppose we wish to *automate* the procedure of opening a data file, extracting numerical data, assigning each array a specific name, and finally plotting everything, in a way that we like (including labeling of the axes). To input (or output) files, we need to learn how to work with "strings" of letters of the sort found in filenames and the "pathname" that tells your computer where to find a file on the hard disk.

The filenames for the data sets that we imported earlier in this tutorial had a base name ("alexa_10s_") followed by an integer running from 1 to the last file of the run (I've only given you 10 files, though 30 or 40 is typical), and then an extension indicating the file format (.SIN). If we are to load all of the data sets you have, it would be hard to keep track of the various types of information stored were we to rely upon the *default* wave-naming scheme ("wave0, wave1, ..."). A better scheme is to use array names that remind us of which file the data came from, so "AC1_0", "AC1_1", and "AC1_2" for the first set, "AC2_0", "AC2_1", and "AC2_2" for the second, and so on. This will be one of the tasks of our macro.



As you've seen in our previous examples, a macro definition starts with something like:

Macro mName(parameter list)

and ends with "End." In between that beginning and the end, you can enter as many variable definitions and commands as you need. Where I've written "mName," you really ought to choose a more specific name that makes sense, for example, LoadDisplay is the

name we chose for this exercise. Where we've written "parameter list," there can be as many as 10 parameters in the list, though here we'll have just one, which we've called "name," an identifier for the file of interest. (As with the name of our macro, you can call your parameters whatever you want as long as what you use doesn't have some other meaning in *Igor*.)

The next line(s) in the macro must define the parameter data type(s). In our case, we'll be passing part of a filename, so it will be a string variable. In the Procedure Window shown above, I've typed onto one line, "String name," to specify that the parameter I've called *name* will be of the data type *string*. You can see that I've also defined the data types for any other local variables to be used in our macro. (These will be used to compose each distinct filename to be loaded and the array names to be stored in *Igor. Pro*.)

You should now do the same, to create the same macro shown in the previous screenshot:

- In your own Procedure Window, following the lines that name our macro and define the data type for our parameter, name, now define the string variables "fName" (for the data *file name*), along with "wName" (for the base name of array names), and "wName0" and "wName1." In *Igor Pro*'s macros, every single variable to be used must be declared.

- Now we're ready to start entering *commands*, starting with some string manipulation, which is really easy in *Igor*: strings can be concatenated with a "+" sign. For example, the code

  fName = "alexa_10s_" + name + ".SIN"

  will result in assigning to fName the string "alexa_10s_1.SIN" when "name" is the string "1", "alexa_10s_2.SIN" when "name" is the string "2" and so on. When we use this macro, setting the input parameter to the string "n" will call up n^{th} data file.

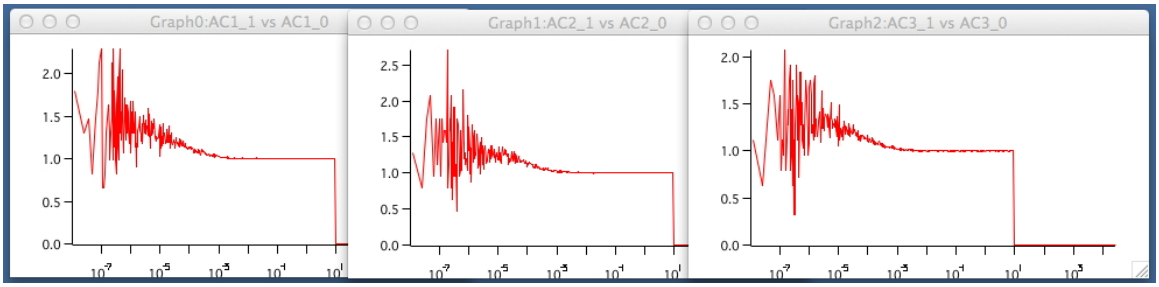
- We do need to tell *Igor* where on the hard disk to find the file (this is called the *path*). Wherever you've stored these datafiles on your computer, the path will undoubtedly, be different than what's shown in the figure above, which is why we told you to copy that down during the previous exercise. You can always ask for help if you have trouble figuring this part out.

- Once the string names, with path, are constructed, use the command that we formed in the last exercise:

  LoadWave/G/D/L={0,16,608,0,3}/A=$wName/O fName

Notice the "$" character before the wName string. This forces a *conversion* from a string to a name of a data array. The filename parameter *should* be a string, so we don't need to convert fName to a string, it is already a string. Details on the particular syntax used here are available under Help → Command Help.

- Next, we add commands so that once the data is imported, *Igor* will plot and set the horizontal axis to a log scale, adjust the plot symbol color, and label the axes.
- Finally, to use the macro, you can simply select it from under the Macro menu! – This is a nice alternative to the approach we used in previous macro and function examples. Of course, it would also work to type into the command window LoadDisplay("1"), and hit enter, or to try LoadDisplay("2"), *etc.*



Truly, *Igor* is now at your command!

9.2 Functions

The difference between a macro and a function is simply that, as the name implies, functions return a *value* (which may be an entire wave).

9.3 Background Information

The following example generates two functions relevant to the birefringent crystals that we use to generate entangled photons for a series of instructional labs in quantum mechanics, for which step-by-step instructions are provided in the Chapter 1 of the senior undergraduate honors thesis of IWU Physics major Andy Ding. Since such quantum information is becoming such an important part of the undergraduate curriculum in physics, we begin with a few words about how light interacts with the particular birefringent crystals that we employ. To understand their use, we first recall that a light wave consists of an oscillatory electric field. (There's also a magnetic field component, but that is smaller by a factor of c, which is a very big factor indeed and, in any case, most optical glasses are non-magnetic and so we usually don't need to concern ourselves too much with the magnetic response of optical components.) When we apply an electric field, E, to a polarizable material, we

get a (*very* slight!!) sloshing of the charges inside the material, which we refer to as the "polarization" of the material. In general, that polarization can be described by a quantity P, whose dependence upon the electric field may be written in terms of a series expansion:

$$P = \epsilon_0(\chi E + \chi_2 E^2 + \chi_3 E^3 + \cdots)$$

In other words, the bigger the electric field, the bigger the sloshing of charge within a polarizable material. This response is dominated by the first-order term, which is to say that χ (called the *linear* susceptibility) is much bigger than χ_2 or χ_3, and for most materials - even when exposed to the intense fields found in laser light - those extra terms are really negligible, ...but for the BBO crystals we use, they are at least big enough to lead to some measurable effects.

The simplest lightwaves we can write down have electric fields of the form $E = E_0 sin(\omega t)$, which yields:

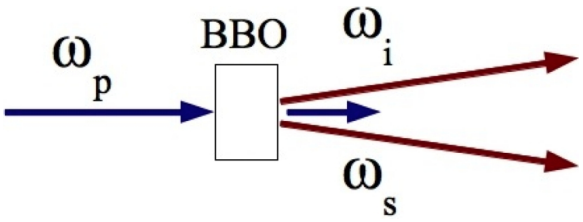
$$P = \epsilon_0(\chi E_0 sin(\omega t) + \chi_2 E_0^2 sin^2(\omega t) + \chi_3 E_0^3 sin^3(\omega t) + \cdots)$$

Use of simple trig identities allows this to be re-written as:

$$P = \epsilon_0(\chi E_0 sin(\omega t) + \chi_2 E_0^2 \frac{1}{2}\left[1 - cos(2\omega t)\right] + \chi_3 E_0^3 \frac{1}{4}\left[3sin(\omega t) - sin(3\omega t)\right] + \cdots)$$

Note that the "linear" response has the same frequency as the incident field, much in the same way that an oscillator that obeys Hooke's Law responds, in steady state, only at the frequency of the drive. However, any non-negligible nonlinearities will result in second harmonic and third harmonic frequencies.

This sort of "nonlinear optics" is key to how we create many of the laser wavelengths of scientific or technical interest, and so is at the heart of *many* opportunities, but is also *key* to all of the single-photon quantum experiments are currently incorporated into the undergraduate instructional labs at many institutions (addressing, *e.g.*, how we produce the entangled photons used for teleportation, quantum computing, disproving local realism, *etc.*) Each of these experiments makes use of a nonlinear effect in which a photon incident on a non-linear optical crystal (BBO) stimulates the emission of two new photons, whose frequencies add up to the frequency of the incident photon, as is shown schematically below:

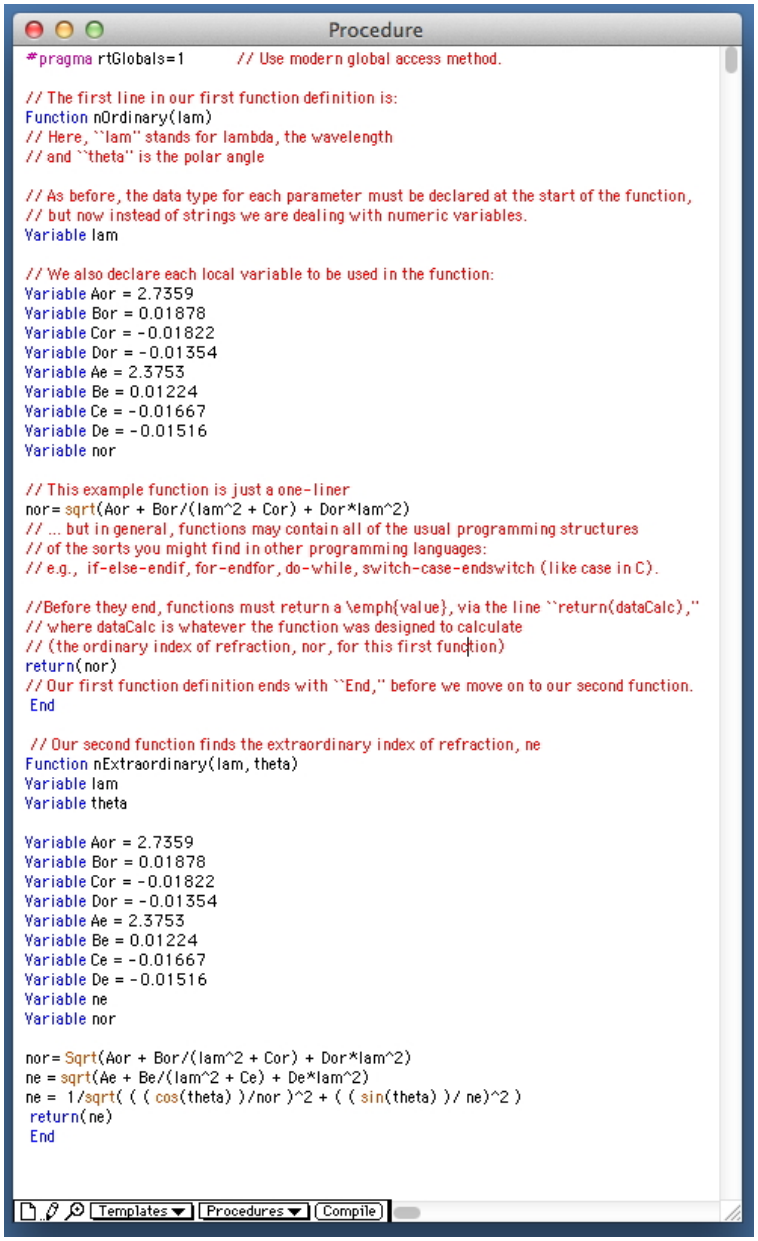


A single 0.405-μm photon can be absorbed and re-emitted as a pair of quantum mechanically *entangled* photons, each with wavelength 0.810 μm. However, this effect will only happen a very tiny fraction of the time at the light intensities in our experiments. Roughly one in 10^{11} photons gets "downconverted," due to a very small nonlinear optical susceptibility (*i.e.*, the χ_2 coefficient in the formula describing the polarization of our BBO crystal; here χ_3 is negligible). Moreover, the effect is highly dependent upon lining up the BBO crystal "just so." The point of this *Igor Prop* exercise is to generate a plot that will tell us just how to hit the BBO crystals with a 0.405-μm pump laser so that the transmitted light might, rarely, consist of as many photons as possible with longer wavelength, 0.810 μm (*i.e.*, we need to calculate the angle-tuned condition for Type I collinear phase matching of parametric downconversion.)

Birefringent crystals are characterized by anisotropy: electrons can more easily slosh along some directions and less easily along others. For BBO crystals, there is a significant difference in how easily the electrons can slosh in the x- or y- directions (yielding what we term the *ordinary* index of refraction) versus how easily they can slosh along the z-direction (yielding what we call the *extraordinary* index of refraction). Suffice it to say that our BBO crystal is cut along a particular direction chosen to serve our purposes. Our first function will calculate the ordinary refractive index for BBO crystals as a function of wavelength (in microns) and our second function will calculate the extraordinary refractive index of BBO as a function of wavelength and polar angle to the optic axis (the direction of laser propagation).

9.4 Intermediate Exercise B: Defining a Function

Open the file BBOPhaseMatch.pxp and you will find the function definitions shown below in the Procedure Window. (Again, when working with code, it is a good idea to expand the window and to zoom in by a factor of two).



In general, once you've typed a function definition into the Procedure Window, *Igor* will *compile* that function as soon as you click in another window. (If there is some syntax error, you will get an alert.) Once compiled, the function can then be called from the Command Window and also from within other function definitions.

Produce the graph shown by entering the following commands into the Command Window:

- make/n=1000 theta, n, nor
- theta= x/1000
- n = nExtraordinary(0.405,theta)
- display n vs theta
- nor = nOrdinary(0.810)
- append nor vs theta
- Label left "Index of Refraction";DelayUpdate
- Label bottom "Angle (rad)"
- TextBox/C/N=text0/F=0/A=MC "n-ordinary"
- TextBox/C/N=text1/F=0/A=MC "n-extraordinary"