



Advanced MATLAB Techniques for Raman Spectral Processing

Matthew Wojciechowicz, Middlebury College and Thayer School of Engineering

*Mentors: Peter Brewer, Ed Peltzer, Peter Walz
Summer 2016*

Keywords: Raman Spectroscopy, MATLAB, Peak Fitting, Marine Jellies

ABSTRACT

Gelatinous creatures are among the most common life forms found in the deep sea, able to maintain structural integrity and ease of fluid motion through an impressive range of temperature and pressure combinations. Upon investigation of both transparent organism bodies (the mesoglea) and the surrounding seawater it became clear that they accomplish this in part through a sophisticated manipulation of the water's molecular structure itself, which was found to be distinct from the structure of gelatin created by land creatures. These investigations were carried out using *in situ* laser Raman spectroscopy at a depth range of 300–2800 m, with additional spectra collected in the lab using a temperature controlled pressure cell to replicate sea conditions.

Qualitative differences in the spectra were easily observed, with our results clearly showing that:

- i) The gelatinous mass effectively excluded salts, with zero sulphate ion being detected.
- ii) The water bending modes are absent from the gelatinous spectra.
- iii) The water stretching modes of the marine creatures are highly modified from the typical five band liquid pentamer structure.

Quantization of these differences remains difficult, although not intractable, using a MATLAB multi step post-processing algorithm. The gelatinous creatures' stretching regime can be decomposed into just three peaks, with the remaining two peaks typically observable in liquid water either sufficiently damped or fully absent in the creatures' bodies.

Comparing to spectra of gelatin derived from bovine sources, there is a similar lack of bending modes, but

also a strong fluorescence which had been completely absent in the jellies’ mesoglea. The lack of fluorescence in the marine gel suggests a very different chemical composition in the gel-forming component that provides maximum flexibility of movement with minimal expenditure of chemical energy.

Rather than using the old processing technique whereby custom scripts were written for each chemical analysis and comparison spectra were taken for baseline subtraction, a generalized spectrum analysis tool with a GUI environment was developed that allows the user to start from raw data and produce fits for up to five gaussian curves.

INTRODUCTION

Raman spectroscopy examines the inelastic scattering of light off a subject of interest, with the shifts in wavelength of the scattered photons containing information about the various physical states of the target. Through examination of the reflected photons, rotational, vibrational, librational, and other states of molecules can be determined and further understood. Because of this, Raman spectroscopy has broad applications across many branches of physics, chemistry, and biology. Recently, this technique was used to determine the hydrogen bond strength in water by using integrated component intensity ratios characterized by the van ‘t Hoff relationship [1], investigate the protein structure of Pacific whiting surimi [2], investigate the effects of temperature on the coupling of stretching modes among water composed of different hydrogen isotopes [3], and even detect brain cancer in humans [4]. This also makes it a prime candidate for investigating the molecular structure of water in marine jellies *in situ*.

The structure of liquid water has been studied intensely, with results varying and many stable cluster structures being found [5]. The most common model used involves a pentamer structure of hydrogen bonded H₂O molecules [1, 3, 5, 6]. This structure has defined vibrational, rotational, and librational modes [1, 3]. These modes can be separated into two main regimes in the 250–4400 cm⁻¹ range, the bending regime at approximately 1200–1900 cm⁻¹ and the stretching regime at approximately 2500–4000 cm⁻¹. The bending band can be separated into two modes while the decomposition of the stretching band is debated, but most commonly is believed to be composed of five modes representing the intra- and inter-molecular vibrations. The effect of temperature on the vibrational modes is minimal, while its effect on the stretching modes is significant and can be modeled as a van ‘t Hoff function. The effect of pressure on the stretching modes is a small monotonic damping, while its effect on the bending modes is even smaller still.

MATERIALS AND METHODS

SAMPLE COLLECTION

The *in situ* experiments were conducted using the MBARI ROVs Ventana and Doc Ricketts aboard the MBARI research vessels Rachel Carson and Western Flyer, respectively. The second iteration of the Deep Ocean Raman In Situ Spectrometer (developed by Kaiser Optics, from here on referred to as DORISS II or

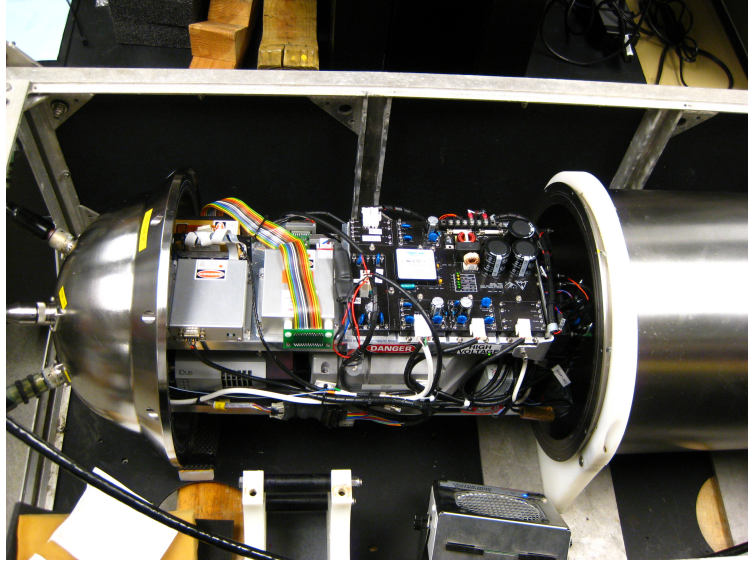
just DORISS) was mounted on the ROVs for the dives, in addition to a 6 L detritus sampler with a custom glass exterior manufactured by Adams & Chittenden, which was used to collect individual cnidarians or ctenophores. Once captured, the sampler was flooded with Argon gas introduced through the lid, causing the jelly to rest immobile on the metal stage. The laser was then focused through the glass wall of the sampler and the focal point adjusted to be inside the gelatinous body of the specimen, at which point spectra were collected. Some of this equipment can be seen in Fig. (1). Overall, 41 jelly spectra (from 12 specimens), 21 background sea water spectra, 4 bovine gelatin spectra, and 1 argon headspace spectrum were collected *in situ*.

Different Argon gas introduction methods were used for each dive, selected according to the maximum operating pressure of the dive. On Ventana, the maximum pressure of approximately 800 dbar absolute meant a simple gas tank system could be used. As the gas tanks can handle pressures of approximately 2000 dbar, it was sufficient to rely on the pressure in the tanks to force gas into the sampler. On Doc Ricketts, however, the maximum operating pressure of approximately 2800 dbar required a different system. As the tanks cannot be pressurized this high, piston accumulators were used instead. The piston accumulators, pictured in Fig. (2), were pressurized to 2000 dbar at the surface. Below 2000 m a hydraulic ram was used to periodically pump seawater into the other end of the piston, allowing the gas to have a small positive gauge pressure on the order of tens of dbar. Once samples were captured, the ram was used to pump additional water into the piston, forcing the gas out the other end and into the D-sampler. Another advantage of this system is the ability to reclaim the gas, reducing the total volume of gas required to collect multiple samples. Once the desired spectra from a sample had been acquired, the hydraulic ram can be operated in reverse, pulling the gas back from the D-sampler into the piston so it can be reused on the next sample.

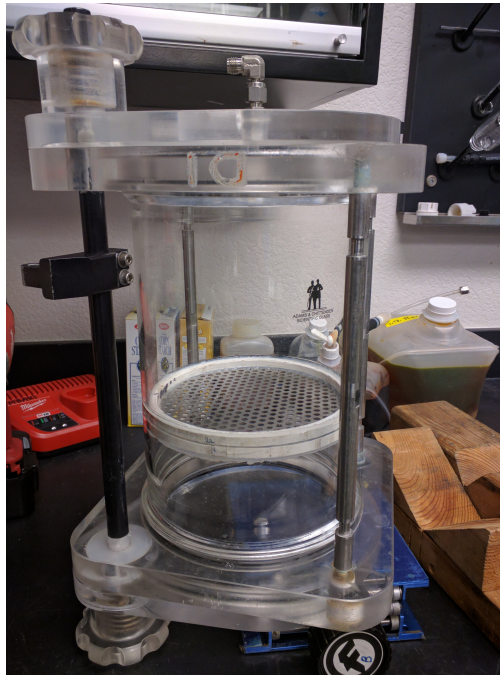
The lab spectra were collected using the DORISS I spectrometer, shown in Fig. (3). These spectra were collected in conditions set to mimic *in situ* conditions by placing sea water in a temperature controlled pressure cell. CTD data was collected from all three dives and average temperatures and pressures calculated in 100 m increments for use in the pressure cell. The temperature was controlled using a Thermo Scientific HAAKE DC10-K10 refrigerated circulating water bath, and the pressure cell was designed by Sam O. Colgate, Inc. In addition to the sea water spectra, several spectra of bovine gelatin, ice, and air were collected using this lab system. In total, 9 pressure cell sea water spectra, 4 air spectra, 3 ice spectra, and 2 bovine gelatin spectra were collected in lab.

DATA PROCESSING

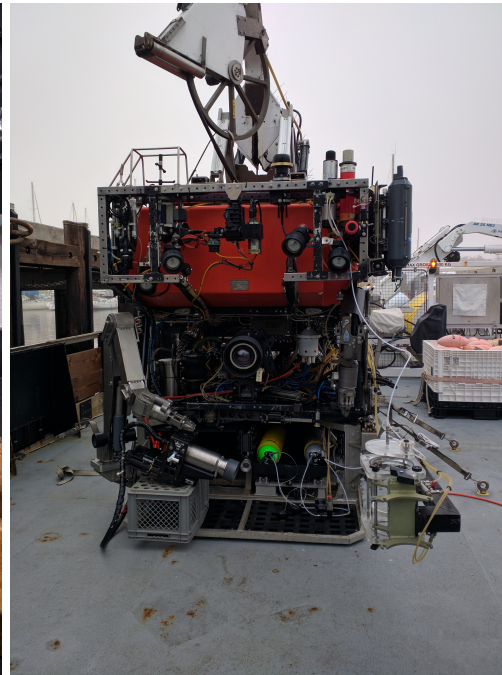
Post-processing of the data was accomplished by developing a generalized GUI Raman spectral analysis tool in MATLAB. Spectra were accumulated into a structure with fields containing the original file name; a vector containing the Raman shifts of the data; a vector containing the relative intensity counts corresponding to each shift; the dive each spectrum was acquired during (if applicable); the type of spectrum (sea water, marine jelly, bovine gelatin, etc.); the species of jelly (if applicable); the depth the sample was acquired at; the surrounding water temperature, salinity, pressure, and oxygen content at the time of sample acquisition (if applicable); which number jelly it was on the dive, and which position on the jelly (if applicable); the shutter exposure time; and any miscellaneous notes relevant for the sample. The GUI then allows the user



(a) DORISS II



(b) Custom D-sampler



(c) ROV Ventana

Figure 1: Various equipment used for spectra collection. a) The DORISS II system partially inserted into its pressure housing. b) The custom Adams & Chittenden glass D-sampler with perforated metal stage. c) The sampler and laser probehead mounted on the ROV Ventana. On the shelf are the two gas tanks used to bring Argon down to flood the sampler, and on the top right are the niskin bottles to collect the samples for lab use.

to go through a series of steps to analyze each spectrum. The steps include:

1. File selection
2. Limit selection and data normalization
3. Data smoothing



Figure 2: Piston accumulators like those used on the Doc Ricketts dive. One end is pressurized at the surface, then seawater is pumped in the other end at depth to create a pressure differential.

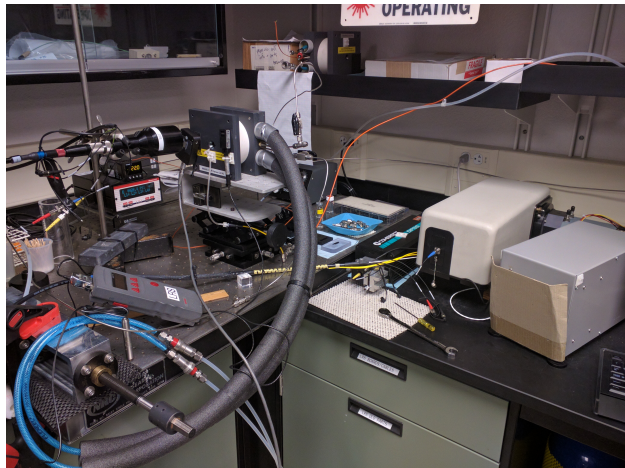


Figure 3: DORISS I system (right) configured to take spectra of temperature controlled seawater in the pressure cell (left)

4. Baseline selection
5. Summed Gaussian fitting

with options for each step and the code itself provided in the appendices.

RESULTS

PROGRAM DEVELOPMENT

The development of the spectrum analysis software went through several phases. Before this project began, the Brewer lab already had a number of MATLAB scripts written for specific applications when analyzing spectra. These often worked by comparing two spectra to extract the relevant information, taking a spectrum of the object of interest and a second spectrum of the relevant background (e.g. a spectrum

of methane dissolved in decane and a spectrum of pure decane) and subtracting the background from the signal, leaving just the relevant information in the final spectrum. This technique is sufficient for analysis such as the methane/decane example, but falls short when investigating the structure of materials that don't have a relevant strong background we can compare to.

This amplifies the problem of dealing with the weak background that will still be present, and how that information can be removed to leave only the signal of interest. In general, thermally broadened spectral lines take the functional form of Gaussians

$$I(\nu) = I_0 e^{-\frac{m_0 c^2 (\nu_0 - \nu)^2}{2kT\nu_0^2}},$$

where I is the light intensity, ν is the wavenumber, m_0 is the mass of the excited object (atom or molecule), c is the speed of light, ν_0 is the wavenumber for the object at rest, k is the Boltzmann constant, and T is the temperature [7]. Because of this, development started with attempts to fit the sum of several Gaussians and a background term to manufactured data. It was discovered that the MATLAB functions included in the curve fitting toolbox for fitting Gaussian curves were insufficient to handle even artificial data, depending too much on the nature of random noise. Instead, a package called `fminspleas` was used for all peak fitting from this point on. `fminspleas` fits a linear combination of non-linear terms to a multivariable scalar function, ideal for fitting spectral data as it allows flexibility not only in the number of peaks fitted, but also can place bounds on input parameters and fit background terms as well.

First attempts were made by attempting to fit several Gaussians plus a constant offset, then a linear offset, and eventually more complicated background functions such as quadratics, cubics, exponentials, logarithms, and others. Although this approach was well equipped to deal with any artificial data (even with large amounts of normally distributed random noise on top), when applied to real spectra it fell short of its goals, unable to return satisfying fits. It was realized this entirely had to do with the baseline issues, particularly the inflexibility of a simple function to define the baseline in addition to the difficulty of using the program in a command-line interface. As a result, the second iteration of the program involved a graphical user interface (GUI) and a custom-defined piecewise baseline.

The GUI feature would allow the user to select what information they desired the baseline to utilize, then see in real time as the baseline is rendered and what effect it has on the corrected spectrum. Initially a cubic `spline` was used to fit the baseline points, but it was discovered that this method resulted in excessive oscillation, an undesirable characteristic for spectroscopy. Instead, the `pchip` function was used, which fits a piecewise cubic Hermite interpolating polynomial to its input. This was more successful, as the `pchip` function has zero overshoot and minimal internal oscillation.

The first iteration of this system brought up a series of windows, each detailing a step of the process. First, a window appeared allowing the user to select which `.asc` file of data to analyze. Next, two windows appeared, one allowing the user to pick a section of the spectrum and the second showing the currently selected window. Then two new windows appeared, one showing the unsmoothed data with an input to decide a smoothing factor and the other showing the data with the selected smoothing and the difference between the smoothed and unsmoothed data. Next another window showed up, in which the user would select their baseline by clicking points on the data that would be used as inputs for `pchip`. Finally, two more

windows appeared, one showing the baseline corrected data and eventually the fitted curves, and the other with a series of inputs for selecting the number of Gaussians to fit, parameter estimates for the Gaussians, and parameter bounds for the Gaussians.

Although this system worked, it had a number of shortcomings. These included:

- It was prohibitively difficult to reproduce previous fits
- There was no mechanism allowing the user to change previous inputs, and a mistaken input would result in having to restart the entire fit
- Difficulty quitting the program in the case of a mistaken input, compounding the previous problem
- No mechanism for saving fits, compounding the first problem
- The flurry of windows appearing and disappearing at each step

because of this, a second iteration of the GUI was developed.

FINAL PROGRAM

The final program streamlined the process, combining all inputs and graph displays into a single window, with minimal other windows appearing throughout the process. The main window, shown in Fig. (4), is where the user interacts with the data, and is stepped through the process of analysis. By saving the raw spectra in a secondary file containing a structure, additional information about each spectrum is stored as well. The fields of the structure allow the user to view information about the spectrum: the type of spectrum, vehicle collected on, exposure times, and by referencing the CTD data environmental conditions such as temperature, pressure, salinity, oxygen content, and others.

At each step, only the relevant inputs can be changed, minimizing the chances of a misclick and allowing the program to check that inputs are valid (lower limits always less than upper limits, and within the range of the data). There is also an opportunity to change previous inputs such as smoothing factors, window limits, and baselines, although these usually reset the later steps to prevent conflicts inside the code. Further details about what variables are reset and which are preserved across changes, see Appendix A. One of the main advantages of this single-window method is that information generated such as baseline points and error functions are easily passed from one function to the next, making it simple to save the steps used in an external file so that fits can be reproduced, an improvement over the previous iteration of the program.

One of the reasons it was difficult to quit the previous iteration had to do with the `waitfor` function used to wait for the user's input. With the multiple window approach, the program pauses until a certain property of the figure window is changed, then returns the currently running function, advancing the user to the next step. However, the way MATLAB implements this function also causes the main function to return if the figure window is closed, meaning an attempt to quit usually resulted in the program being advanced to the next step instead. With the single window approach, there is no information passed from one window to the next, so closing the window quits the program instead. It also makes analyzing multiple spectra in a row more streamlined. In the previous iteration, each new analysis would require a separate run of the function, but now the user can return to the file selection step without quitting the program, and continue looking at new spectra inside the same function run.

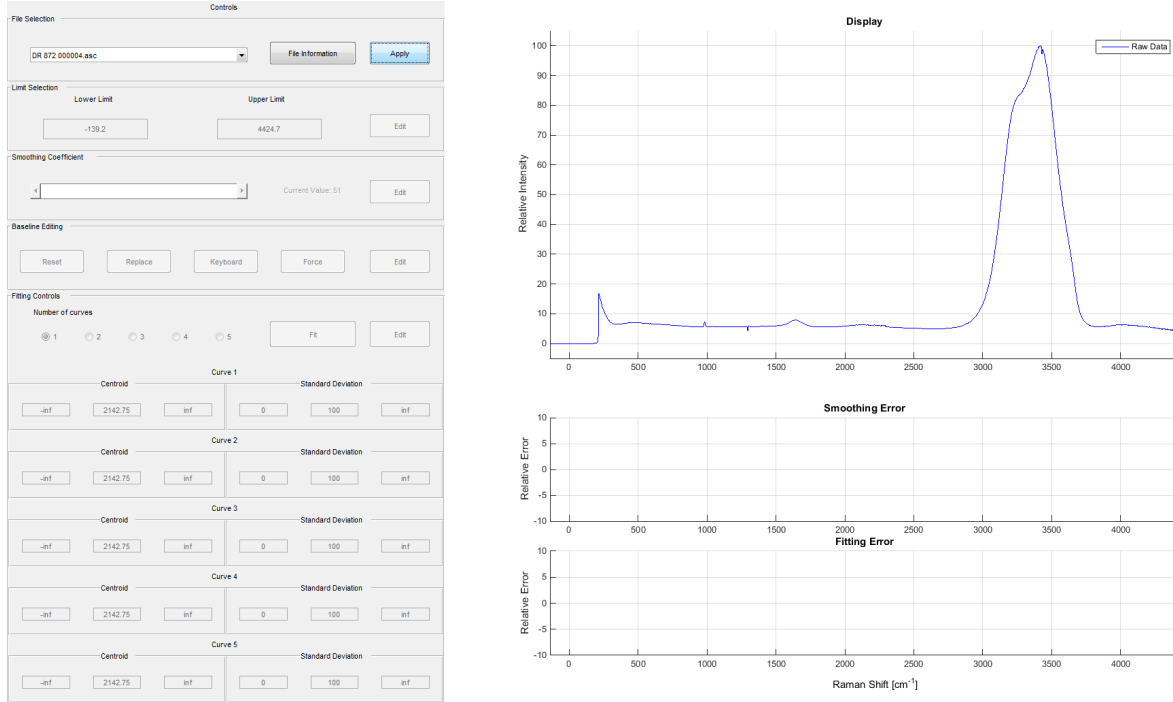


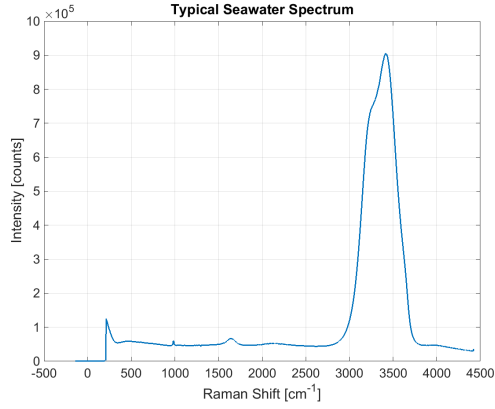
Figure 4: The user interface for the final program. Each step of the process is input on the left side of the window, with the results of the step pictured on the right. Additional windows appear if the user attempts to do something not allowed (e.g. a lower bound greater than an upper bound), and upon fitting to display the resulting parameters.

QUALITATIVE RESULTS

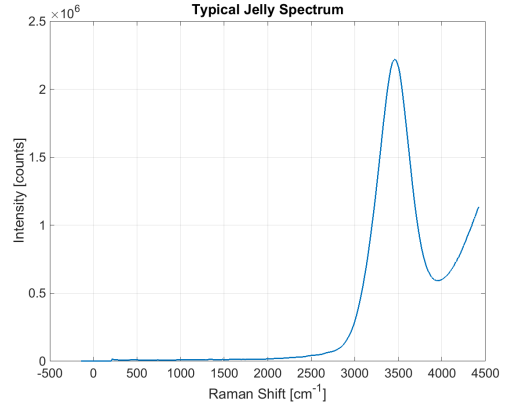
After collecting the spectra there were a number of qualitative results that were immediately apparent, all of which can be observed in Fig. (5). One of the more striking results is the general lack of bending modes in solidified water, regardless of solidification method. None of the jelly, gelatin, or ice spectra have any peaks around 1600 cm^{-1} , suggesting that when placed in a lattice structure H_2O molecules can no longer bend, irrespective of the specific shape of the lattice.

The bovine gelatin has a broad fluorescence throughout the entire range of sampled wavenumbers, which is to be expected when examining animal specimens, as this range of fluorescence is typically caused by the aromatic amino acids (Phenylalanine, Tyrosine, Tryptophan, Histidine, etc.). This same fluorescence, however, is absent in our jelly spectra. It is possible that the energy states get shifted to higher wavenumbers, and the fluorescence seen around 4000 cm^{-1} is the beginning of the peak, but more likely it suggests a drastic compositional change in the protein structures in jellies. Some possibilities include proteins that are deficient in the aromatic amino acids, glycoproteins, or carbohydrates, but more investigation is necessary to reach any conclusions. The lack of aromatic amino acids in jellies is supported by findings that *Hydra vulgaris*, classified in the same phylum as the samples presented here, are lacking in the aromatics [8].

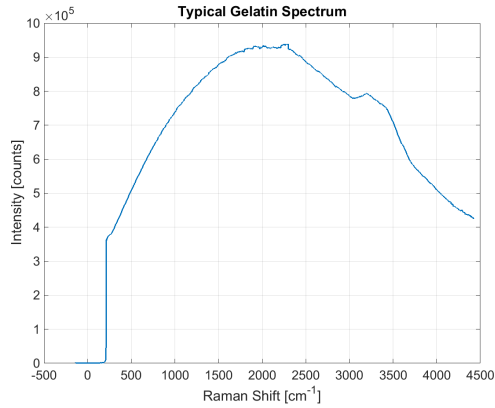
Both the jelly and ice spectra readily show changes in the structure of the water stretching band (approximately $2500\text{--}4000\text{ cm}^{-1}$), while in the raw data for the bovine gelatin it is unclear whether any significant change can be observed. Further analysis of the spectra using MATLAB makes this clearer.



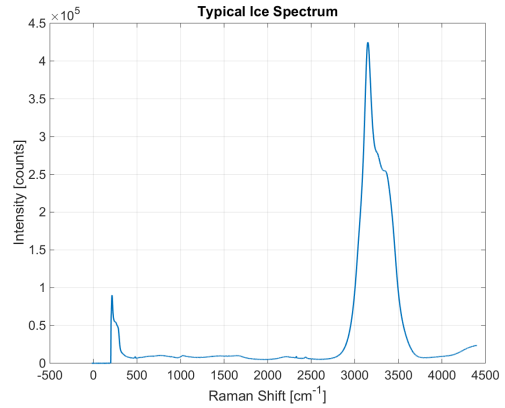
(a) Sea water



(b) Jelly



(c) Bovine gelatin



(d) Ice

Figure 5: Typical examples of the four main types of spectra collected. a) A typical seawater spectrum. Note the bending doublet around 1600 cm^{-1} and sulfate peak just below 1000 cm^{-1} . b) A typical marine jelly spectrum. Note the lack of bending doublet and sulfate peak, modified stretching band around $2500\text{--}4000\text{ cm}^{-1}$, and fluorescence above 4000 cm^{-1} . c) A typical bovine gelatin spectrum. Note the lack of bending modes, possibly modified stretching modes, and broad fluorescence. d) A typical ice spectrum. Note the lack of bending modes and modified stretching band.

Table I: A comparison of the peaks in the water stretching regime for a jelly (b) and the water around it (a), shown graphically in Fig. (6). Note the different number of peaks and lack of shared means.

(a) Sea water					(b) Jelly				
Peak	Amplitude [arb]	Mean cm^{-1}	STD cm^{-1}	FWHM cm^{-1}	Peak	Amplitude [arb]	Mean cm^{-1}	STD cm^{-1}	FWHM cm^{-1}
1	4.78	3049.0	113.7	267.7	1	40.37	3323.5	169.7	399.5
2	77.06	3248.3	102.5	241.4	2	55.91	3489.6	128.5	302.5
3	58.63	3405.2	64.8	152.6	3	9.77	3686.5	102.4	241.0
4	50.25	3496.0	60.7	143.0					
5	25.87	3607.4	57.5	135.5					

The jellies also exhibit some amount of salt exclusions, evidenced by the lack of sulfate peak in the spectra. In liquid water, the normalized amplitude of the sulfate peak can be used as a rough proxy for total salinity, as the sulfate concentration has a well-defined relationship with the total salinity of the water [9]. If this same relation between sulfate peak amplitude, sulfate concentration and total salinity holds in jellies, these results suggest that jellies are able to exclude salts and maintain an osmotic gradient with the surrounding seawater. It is also possible, although unlikely, that jellies have a mechanism that excludes just the sulfate and not the other salts commonly found in sea water.

QUANTITATIVE RESULTS

MATLAB analysis of the data shows that the stretching band of water is highly modified inside the jellies, as shown in Fig. (6). Fits were accomplished to all of the sea water spectra using five peaks, as both the minimal number of peaks required to produce satisfactory fits and as suggested by literature [1]. These peaks were in satisfactory agreement with the values found by others, giving further confidence in the ability of the software to accurately determine the component peaks in *in situ* spectra. By contrast, the minimal number of peaks required to produce satisfactory fits in the jelly stretching regime was just three, suggesting either a significant or complete damping of these energy states in the water structure.

It is apparent from the results shown in Table I that the peak means have no agreement between the jellies and sea water, making it difficult to determine which peaks are the modified stretching modes from seawater, how those modes have been modified, and which modes are absent in the jellies. It does, however, show that not only are there two modes of water stretching that are absent from the jellies, but also that the energies of the remaining states are shifted.

Because of the high level of fluorescence in the bovine gelatin spectra, even after processing the data there is a low signal to noise ratio, with a large amount of oscillation in the data that the fit algorithm was not able to handle properly. Therefore, there are no numeric results from the bovine gelatin, but some additional qualitative results were noted after processing. The general shape of the processed spectrum is more similar to the sea water spectra than the jelly spectra, suggesting the method used to create the gel is distinct from that used by the jellies and may have a less significant effect on the structure of the water, possibly even maintaining the same pentamer structure as that found in liquid water.

The ice spectra collected also did not yield any satisfactory fits, but for different reasons than the bovine

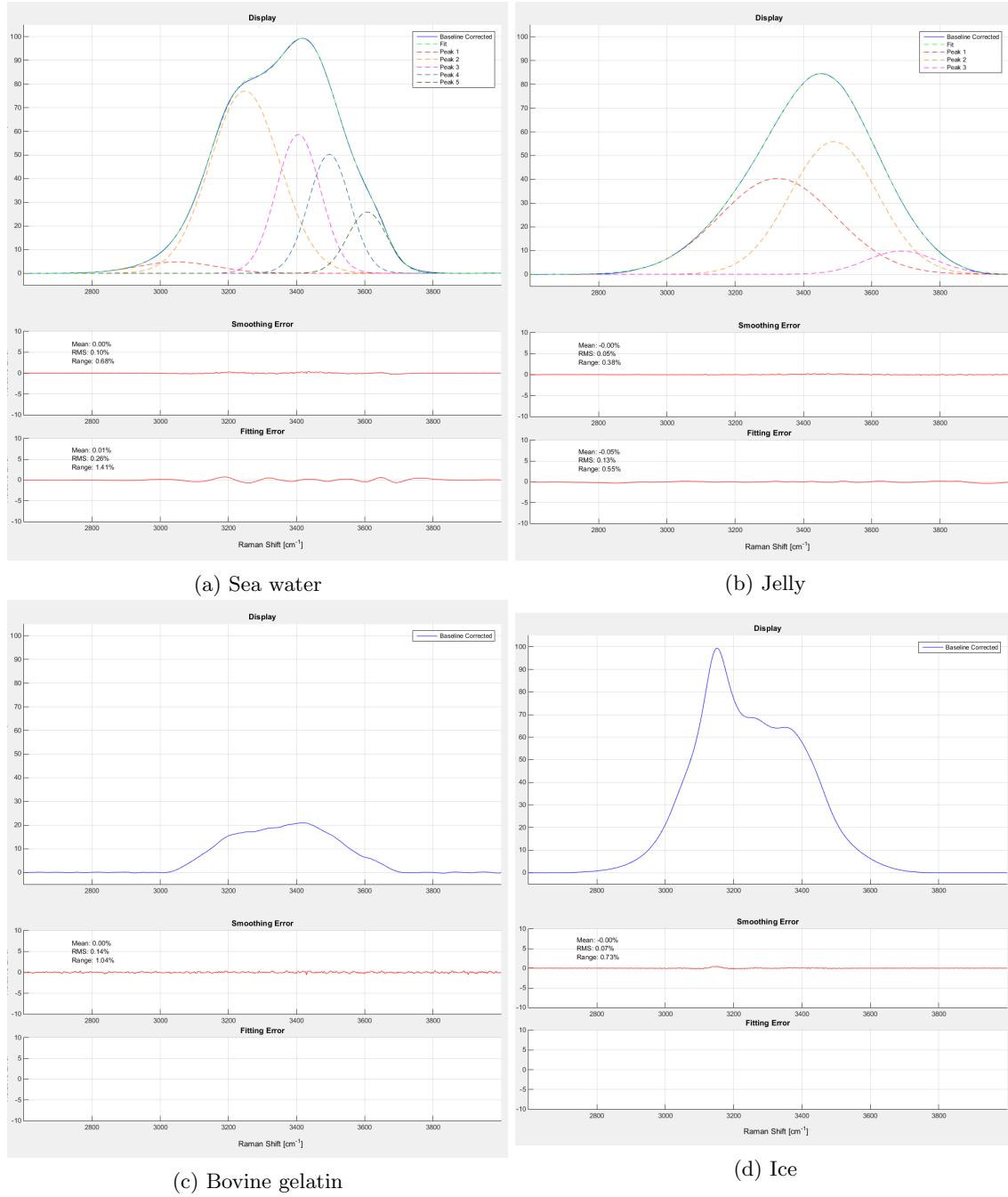


Figure 6: A comparison of the stretching modes found in sea water, marine jellies, bovine gelatin, and ice. a) The typical five modes found in the sea water stretching band. b) The same stretching modes in a marine jelly. Note the different number of peaks, different relative amplitudes, and lack of shared centroids between any peaks. c) The stretching modes in jello. The high fluorescence made actual peak fitting prohibitively difficult, but note that it resembles the sea water more than the jelly regime. d) The stretching modes in ice. No fits were found for this regime that appeared physically plausible, but from the number of local maxima apparent in the graph there are at least three stretching modes, and most likely four or five.

gelatin. Fits were found with both four or five peaks, but the error functions produced had definite shape to them, suggesting that although numerically the summed gaussians were close to the data, any theory to

justify those peaks would be insufficient in justifying the mismatch between the fit and the data. Although, closer analysis of the stretching regime did reveal some additional information, as Fig. (6) shows more clearly that the entire regime is shifted to the left from the sea water data. This makes sense, as a left shift corresponds to a lowering in the energy of the states, and lower temperatures result in lower energies of the states involved. It was also noticed that the relative amplitudes of the local maxima for the higher energy part of the band are similar between the ice and sea water spectra, there is the single sharper peak in the ice around 3150 cm^{-1} that is not apparent in the sea water. This suggests that the relative amplitudes of the various stretching modes are affected more by the phase change than they are by changes in temperature inside the same phase.

DISCUSSION

SUCSESSES

These results show that it is possible to probe the inner chemistry of transparent sea creatures using laser Raman spectroscopy, and come to conclusions about not only their chemical composition, but even their molecular configuration. Satisfactory results were found using peak fitting processing techniques detailing many of the differences between the structure of the water molecules in sea water and in marine jellies. This technique is not perfect, as it could not handle the complications of fitting either the bovine gelatin or ice spectra, but makes significant improvements over the methods used previously.

SHORTCOMINGS

There are several potential issues with the series of steps used to analyze the data; some complaints are easily resolvable while others may require further code development to resolve and/or limit the applicability of the program. A few of these are enumerated here, with further explanation and sometimes potential resolutions in Appendix B.

- In normalization, mapping the intensities to the interval $[0, 100]$ may distort the relative amplitudes
- In smoothing, the behavior of the smoothing algorithm behaves differently around interval ends
- In smoothing, the implicit assumption of equally spaced sample points
- In smoothing, mild attenuation of peaks
- In baseline creation, creation of negative intensity values
- In baseline creation, significant distortion of short, broad peaks
- In baseline creation, extrema of subintervals are the endpoints
- In peak fitting, numeric minima are not necessarily theoretically motivated

CONCLUSIONS/RECOMMENDATIONS

PROGRAM APPLICABILITY

Use of this technique simplifies and streamlines the process of analyzing Raman spectra, requiring almost no knowledge of MATLAB code to run and producing fit data for the spectra in addition to presenting these results in a manner that is easily reproducible or able to be applied to later analyses. There are some subtleties in using the program that suggest a user should familiarize themselves with use before assuming any results they get are physically motivated.

Once a user is familiarized, the program can be used to successfully analyze any Raman spectrum, and in fact any spectrum where the underlying physics suggests Gaussian distributions. The combination of steps involved is a powerful tool for isolating a section of a spectrum and determining its components.

FUTURE WORK

There are two main sections of future work to note, the first is work on the program itself, the second is work to expand and clarify the results found using the program.

For the program, the saving mechanism should be improved, as the current formatting of the saved image isn't ideal. MATLAB has some subtleties when exporting graphics that result in the saved image not appearing exactly how the figure looks on the screen. Although the formatting of the window prior to saving is satisfactory, this export process results in some mismatching between font sizing and axes scaling that make the saved figures not as easy to decipher as desired. Improvements could also be made in the structure method of storing the spectra. At the moment, the user must be familiar working with structures in MATLAB, and there is an extra step involved before spectra can be analyzed. To get more immediate results, a second, scaled down version of the program could be created, that reads in .asc files directly rather than needing the extra step of porting the data to a structure. This would make analyzing spectra immediately after collection easier, to check if data is satisfactory before recovering the ROV, or if a new spectrum should be acquired before releasing the sample.

When it comes to the results, more investigation and more sample collection is necessary before generalized conclusions about marine jellies can be made. The samples collected here were from only three different species of jellies, with only one specimen for two of those species. The program and technique would still be applicable to samples with tissue that is not transparent, although research into the success of Raman spectroscopy for investigating more opaque samples should be carried out first.

This program has wide applicability to studying the effects of several variables on the structure of jellies, and two areas that warrant further investigation are the effects of temperature and pressure on the various modes of water. The effects on water of these variables have been studied in detail [1, 5, 6, 9], but as yet no analogous study has been done on jellies or other marine animals.

The results found about the difference between bovine gelatin and marine jellies are intriguing, and more

study can be done to see if other gel forming methods use similar or novel approaches. Agars derived from kelp would be of particular interest, as they are another source of gel found in the sea, and it would be worthwhile to see if jellies are unique in their gelling methods or if other sea creatures use similar methods.

ACKNOWLEDGEMENTS

Thank you to Peter Brewer for guidance and ample reading material to bring me up to speed on the basics of marine chemistry and the behavior of liquid water structures, assistance on the physical interpretation of my results, and plenty of stories of the development of the scientific world over the past 50 years; to Ed Peltzer for indulging my continuously changing code, fantastical ideas of turning commandline scripts into a GUI, and additional physical interpretation of what exactly I was discovering; to Peter Walz for a crash course in submarine system design and the operation of the laser Raman systems; to Jenny and Jeff Paduan for housing and feeding me this summer, and showing me what life in California can be like; to George Matsumoto and Linda Kuhn for helping the transition into working at MBARI; to the other interns for a number of adventures and fun experiences over the summer; and to MBARI and The Packard Foundation for funding and organizing my internship.

References

- [1] Carey, D. M., & Korenowski, G. M. (1998). Measurement of the Raman spectrum of liquid water. *The Journal of Chemical Physics*, 108, 2669–75. <http://dx.doi.org/10.1063/1.475659>
- [2] Bouraoui, M., Nakai, S., & Li-Chan, E. (1997). In situ investigation of protein structure in Pacific whiting surimi and gels using Raman spectroscopy. *Food Research International*, 30, 65–72. [http://dx.doi.org/10.1016/S0963-9969\(97\)00020-3](http://dx.doi.org/10.1016/S0963-9969(97)00020-3)
- [3] Hare, D. E., & Sorensen, C. M. (1992). Interoscillator coupling effects on the OH stretching band of liquid water. *The Journal of Chemical Physics*, 96, 13–22. <http://dx.doi.org/10.1063/1.462852>
- [4] Jermyn, M., Mok, K., Mercier, J., Desroches, J., Pickette, J., Saint-Arnaud, K., Bernstein, L., Guiot, M.-C., Petrecca, K., & Leblond, F. (2015). Intraoperative brain cancer detection with Raman spectroscopy in humans. *Science Translational Medicine*, 7, 274. <http://dx.doi.org/10.1126/scitranslmed.aaa2384>
- [5] Walrafen, G. E. (1964). Raman Spectral Studies of Water Structure. *The Journal of Chemical Physics*, 40, 3249–3256. <http://dx.doi.org/10.1063/1.1724992>
- [6] Walrafen, G. E., Hokmabadi, M. S., & Yang, W. H. (1986). Raman isosbestic points from liquid water. *The Journal of Chemical Physics*, 85, 6964–9. <http://dx.doi.org/10.1063/1.451383>
- [7] Haken, H., Wolf, H. C., & Brewer, W. D. (1996). *The Physics of Atoms and Quanta: Introduction to Experiments and Theory*. Springer-Verlag, 5th edition.
- [8] Jr., M. P. S., Madden, M. E., Zhang, X., Gunwar, S., Huff, J. K., & Hudson, B. G. (1991). Extracellular Matrix (Mesoglea) of *Hydra vulgaris*. *Developmental Biology*, 148, 481–494. [http://dx.doi.org/10.1016/0012-1606\(91\)90266-6](http://dx.doi.org/10.1016/0012-1606(91)90266-6)
- [9] Dolenko, T., Burikov, S., Sabirov, A., & Fadeev, V. (2011). Remote Determination of Temperature and Salinity in Presence of Dissolved Organic Matter in Natural Waters Using Laser Spectroscopy. *EARSeL eProceedings*, 10, 159–165.

APPENDICES

A. Code Description

File Selection

In the file selection step, the user is provided with a dropdown menu displaying the source filename for each spectrum in the data structure and a button ‘File Information.’ The file information button brings up a new window displaying the file information entered in the various other fields of the structure. This option is available throughout the analysis.

Selecting a new file clears all window limit, smoothing, baseline, and fit parameter information that may have been entered.

Window Selection

In the window selection step, the user can select upper and lower limits for the section of the spectrum they would like to further analyze. The program only allows the user to select window limits where the lower limit is less than the upper limit and both limits fall within the data range of the original file. As each limit is selected, the data is normalized to fall on the interval $[0, 100]$. Normalization is a linear function of each intensity that maps the highest intensity value to 100 and the lowest intensity value to 0.

Selecting new limits clears all smoothing, baseline, and fit parameter information that may have been entered, but preserves the selected file.

Smoothing

In the smoothing step, the user selects the size of the smoothing window (in units of data points). The smoothing window can take on odd integer values from 1 to 101, with a value of 5 corresponding to the smoothed intensity value at each point equaling the arithmetic mean of the original intensity value and the two closest intensity values on either side of it. Smoothing is accomplished using the MATLAB function `smooth` which is a part of the curve fitting toolbox.

Selecting a new smoothing value clears all baseline information that may have been entered, but preserves the selected file, window limits, and any fit parameter information.

Baseline

In the baseline creation step, the user creates a baseline function to be subtracted from the smoothed data before fitting peaks. To prevent divergent results around the edges of the data, the first and last point of the interval are always part of the baseline fitting. Once this mode is entered, left clicking on the plotting axes finds the horizontally closest data point to the click and adds it as a fit point for the baseline. Right clicking finds the horizontally closest baseline fit point and removes it from the fitting algorithm, unless that point is a window endpoint. Clicking the ‘Replace’ button puts the most recently removed (by right clicking) point back into the fitting algorithm. All removed points are stored iteratively, so multiple removed points can be replaced if desired. Clicking the ‘Keyboard’ button brings up a text box that lets the user input a shift value and adds the horizontally closest point to that shift value to the fit algorithm. Clicking the ‘Force’

button allows the user to add a fit point that is not also a data point to the baseline fit algorithm. The program adds the exact coordinates of the click to the fit algorithm. Clicking the 'Reset' button resets the baseline to contain only the window endpoints, and also clears any removed points in the process. Baseline fitting is done by entering the points into the MATLAB `pchip` function.

Editing the baseline fit points preserves all other information the user has input.

Peak Fitting

In the peak fitting step, the user chooses the number of gaussian curves they would like to fit to the now baseline-corrected data, anywhere from one to five. For each curve the user can edit the estimated values in addition to the upper and lower bounds for the mean and standard deviation parameters of each fit. The linear preexponential factor for each term is estimated internally by the fit algorithm. Once the desired parameters are entered, the user presses the 'Fit' button to execute a fit.

Once the fit is completed, a new window appears displaying the parameters of each curve in the fit and a button offering the user the option of saving the session so far. If the user chooses to save the session, a save window appears allowing the user to choose the location and file name of the save. The information is saved as a color postscript file with the three fitting axes to the right, and the file information, processing parameters, and fit result parameters displayed to the left.

Editing the fit parameters preserves all other information the user has input.

B. Shortcomings Elaborated

Normalization

In the data normalization involved in the window selection step, normalization is implemented by linearly mapping every intensity value to a space where the highest value in the selected interval is mapped to 100 and the lowest intensity value is mapped to 0. This involves two steps, a constant offset to every point and a constant scaling of every point, and each step can be justified individually. The constant offset at each point can be justified by pointing out that the later baseline correction step introduces a more sophisticated offset for each point, and we can think of this offset as adding to that one, so it introduces no further limitations on the program than those inherent in baseline correction.

The multiplicative scalar mapping is a bit more complicated to justify, as it is introduced mostly as a tool to make data visualization easier. One must use caution when comparing results from different spectra using this algorithm though, as relative intensities are only scaled with respect to the individual spectrum, and peak amplitudes between spectra are not equal (i.e. a peak with amplitude 0.5 in one spectrum is not necessarily equivalent to a peak with amplitude 0.5 in a different spectrum). As such, absolute intensities should not be compared between fits. We are, however, primarily interested in amplitude or area ratios of peaks inside a spectrum, and those are okay to compare (i.e. if one finds that the amplitude ratio of the two bending modes in one spectrum is 1.5 and the same ratio in a different spectrum is 1.4, it is fair to say that the relative intensity of each peak changes due to whatever conditions are different between the spectra). As the Raman shift values are never changed, peak widths are also comparable between spectra, and since the area under a peak is a linear function of the amplitude and width these comparisons are also still valid, in addition to comparisons of peak area ratios between spectra. Should a user desire to deactivate the normalization, they need only change the `normalize` nested function to return `ydata` unmodified and adjust the `YLim` properties of the three axes accordingly.

Endpoints

The smoothing algorithm behavior around the endpoints of a window involves reducing the averaging box (e.g. the endpoints are unchanged, the second and penultimate points only average the point and the adjacents, the third and penultimate points only average the point and the two points to either side, etc.). Accordingly, the user must be sure to select a window that includes some tails that do not interfere with the peaks of interest, which is good practice when defining a baseline anyway. To avoid these issues, the user can use a smoothing radius of 1, which effectively leaves the data unsmoothed.

Boxcar Averaging

Because the smoothing uses a simple moving boxcar averaging method, it implicitly assumes equally spaced data points. This is mostly to reduce computation time, as any other method tried cannot update in real time. In general, spectrographic information will be acquired in samples that are equally spaced, or close enough to equally spaced that this assumption is valid, but if for any reason the user needs to analyze data points that are not equally spaced, inputting the Raman Shift information to the smoothing algorithm may be necessary. See the following subsection for further details.

Attenuation

Again an artifact of the boxcar smoothing, there is some small attenuation of peaks, and the narrower the peak the worse the attenuation. The simplest solution here is to use a small smoothing radius, but if the user desires to smooth more without as much attenuation they should reference the documentation for the MATLAB function `smooth` and adjust the `reloadSmoothing` nested function accordingly. Overall we found that the attenuation was very small and shouldn't interfere with fitting, but the more advanced smoothing techniques would attenuate less at the expense of computation time.

Negative Intensities

When placing fit points to create the baseline it is easily possible to unintentionally create negative, or even highly negative, intensity values. Negative intensity values cannot be physical in origin, and may end up impacting the curve fitting step. It can cause peaks to have a smaller overall amplitude, or even cause peaks to have negative amplitudes, which again cannot be physical.

Peak Distortion

When baselines are overfit, such as adding fit points very close to the peaks, the resulting baseline corrected data can end up reduced in intensity across the board. Although this usually has little to no impact on the larger amplitude peaks, any peak that is expected to be short and broad, such as the bending peak around 1580 cm^{-1} in water, can be narrowed and even shifted by this change in intensity values.

Local Extrema

One of the consequences of using an interpolating function using the Hermite polynomials is that the endpoints of any subinterval are the extrema of the interval. This is desirable in one regard for baseline fitting, as it is part of what helps minimize oscillation of the interpolating function. However it also results in some undesirable behavior in cases such as the bovine gelatin spectra, where the raw data has peaks that are known to be background but cannot be easily defined without local extrema inside an interval. The 'force' option is in part intended to help with this, as it allows the user to place local extrema off the data. The user should be very careful in using this option though, as it can result in false signals.

fminspelas

As is the case with any fit algorithm, all that a program can do is give results that represent numeric minima in error functions between the fitted function and the data. These results are therefore not motivated by theory, and may even be grossly different from the actual underlying physical phenomena. The algorithm used in this program is a direct simplex search augmented to minimize the number of function runs by incorporating a least-squares regression for the linear coefficients of terms. This algorithm is implemented by a MATLAB package called `fminspelas` (available for free on MATLAB central [here](#))

C. Code

```
function AnalyzeNorm()
% Matthew Wojciechowicz
% Comprehensive GUI Raman spectrum processing tool with normalization
%
% 'data.mat' must be a file in the working directory containing structure
% 'data' with the following fields
%
%     File           (name of file source for spectral data)
%     Shift          (vector containing the Raman shift data)
%     Counts         (vector containing the intensity count data, same size as Shift)
%     Dive           (the vehicle and dive number of the spectrum)
%     Type           (type of sample)
%     Species        (species of sample)
%     Depth          (depth of sample, in meters)
%     Temperature    (temperature of sample, in degrees Celsius)
%     Salinity       (salinity of surrounding water, practical salinity scale 1978)
%     Pressure       (pressure of surrounding water, in absolute dbar)
%     Oxygen         (oxygen content of surrounding water, in mL/L)
%     Jelly          (what number jelly it was from the dive)
%     Position       (what number position on said jelly)
%     Notes          (miscellaneous notes about spectrum)
%     Exposure       (exposure time during collection, in seconds)
%
% Only the 'Shift' and 'Counts' fields must be non-empty.
%
% The function loads the file 'data.mat' containing the structure 'data'
% and brings up a GUI allowing the user to do the following:
%
%     Select File:    Browse the spectra contained in the structure,
%                    selected based on the name of the source file.
%
%                    Press the 'File Information' button to bring up a
%                    window displaying the spectrum information contained
%                    in the structure.
%
%                    Selecting a new file clears any limit, smoothing,
%                    baseline, and fit parameter information.
%
%     Select Limits:  Choose a section of the spectrum to investigate.
%                    At this stage the program normalizes the data such
%                    that the largest intensity value is 1 and the
%                    smallest intensity value is 0. Normalization is
%                    a linear function of each intensity value.
%
%                    Editing the limits clears any smoothing, baseline,
%                    and fit parameter information, but preserves the
%                    selected file.
%
%     Smooth Data:    Smooth the selected section of the spectrum.
%                    Smoothing assumes evenly distributed sample points,
%                    done with a moving boxcar average.
%
%                    The value displayed indicates the total number of
%                    points averaged in the boxcar.
%
%                    Editing the smoothing radius clears any baseline
%                    information, but preserves the selected file,
%                    window limits, and fit parameters.
%
%     Apply Baseline: Apply a baseline function to the smoothed data.
```

```

%           The baseline is fit using a piecewise cubic Hermite
%           interpolating polynomial.
%
%           Add new points to the baseline by left clicking,
%           the program uses the closest smoothed point to
%           the user click as a new function fit point.
%
%           Remove fit points by right clicking, the program
%           removes the closest fit point to the user click.
%
%           Replace removed points with the 'Replace' button,
%           the program replaces the most recently removed point.
%           All points removed since last baseline reset are
%           stored, so repeated replacement is possible.
%
%           To prevent divergent baselines, the endpoints are
%           automatically included and cannot be removed.
%
%           Reset the baseline to just the endpoints with the
%           'Reset' button. A confirmation dialog appears in
%           case of accidental press.
%
%           Press 'Keyboard' to add a point based on its
%           x-coordinate. The program adds the point with the
%           x-value closest to that entered. Useful for
%           reproducing fits from a saved file.
%
%           Press 'Force' to allow placement of a baseline
%           fitpoint that doesn't fall on the data. Because of
%           the nature of a pchip fit, there are no internal
%           extrema on any given interval, so if the user
%           believes there is a more complicated underlying
%           baseline this feature allows the user to realize it.
%
%           Editing the baseline preserves all other information.
%
%   Fit Curves:   Fit from one to five Gaussian curves to the baseline
%                 corrected data.
%
%                 Upon executing a fit, a new window appears displaying
%                 the fit results. Pressing the 'Save Information'
%                 button prints the fitted curve, error functions, file
%                 information, fit information, and all parameters
%                 necessary to reproduce the results to a figure, then
%                 lets the user select a file name and location to save
%                 the figure as a color postscript image.
%
%                 Fitting uses the fminspelas direct search algorithm.
%
%                 Editing the fit parameters preserves all other
%                 information.
%
%   See also SMOOTH, PCHIP, FMINSPLEAS
%
%   fminspelas can be downloaded for free <a href = "matlab:
%   web('https://www.mathworks.com/matlabcentral/fileexchange/10093-fminspelas')">here</a>
%   smooth is a part of the <a href = "matlab:
%   web('http://www.mathworks.com/products/curvefitting/')">curve fitting toolbox</a>
%
%% Load data and check 'Shift' and 'Counts' are non-empty and the same size
load data.mat
if any(cellfun(@isempty, {data.Shift}))
    error('''Shift'' field cannot contain any empty entries')
elseif any(cellfun(@isempty, {data.Counts}))
    error('''Counts'' field cannot contain any empty entries')
elseif ~isequal(cellfun(@size, {data.Shift}, 'UniformOutput', false),...
    cellfun(@size, {data.Counts}, 'UniformOutput', false))

```

```

    error(''Counts'' and ''Shift'' entries must be the same size')
end

%% GUI data structure
gui = struct(...
    'shift',struct('orig',[],'clip',[]),...
    'count',struct('orig',[],'clip',[],'smth',[],'base',[],'fitd',[]),...
    'error',struct('smth',[],'fitd',[]),...
    'bslin',struct('pnts',[],'base',[],'rmvd',[]),...
    'limts',struct('lowr',[],'uppr',[]),...
    'smthr', [],...
    'fbnds',struct('lowr',[],'estm',[],'uppr',[]),...
    'frslt',struct('lpar',[],'nlpr',[],'crvs',{[],[],[],[],[[]]}));

% fit tracking
gaussian = @(mu, sig, x) exp(-(x - mu).^2/(2*sig^2));
funlist = {...
    @(c, shift) gaussian(c(1), c(2), shift),...
    @(c, shift) gaussian(c(3), c(4), shift),...
    @(c, shift) gaussian(c(5), c(6), shift),...
    @(c, shift) gaussian(c(7), c(8), shift),...
    @(c, shift) gaussian(c(9), c(10), shift)};

%% Figures
% main figure
h.fig = figure(...
    'Name', 'Spectrum Analysis',...
    'NumberTitle', 'off',...
    'Units', 'normalized',...
    'MenuBar', 'none',...
    'ToolBar', 'none',...
    'OuterPosition', [0.02 0.05 0.96 0.93],...
    'Resize', 'off',...
    'Visible', 'off');

%% Main display
% working axes
h.ax(1) = axes(...
    'Parent', h.fig,...
    'Units', 'normalized',...
    'OuterPosition', [0.35 0.43 0.65 0.57],...
    'XGrid', 'on',...
    'YGrid', 'on',...
    'YLim', [-5, 105]);
h.ax(1).Title.String = 'Display';
h.ax(1).YLabel.String = 'Relative Intensity';
% raw data line
h.line(1) = line(...
    'Parent', h.ax(1),...
    'Color', [0 0 1],...
    'DisplayName', 'Raw Data');
% selection window line
h.line(2) = line(...
    'Parent', h.ax(1),...
    'Visible', 'off',...
    'Color', [0 0 1],...
    'DisplayName', 'Selected Window');
% smoothed selection window line
h.line(3) = line(...
    'Parent', h.ax(1),...
    'Visible', 'off',...
    'Color', [0 0 1],...
    'DisplayName', 'Smoothed');
% points defining baseline
h.line(5) = line(...
    'Parent', h.ax(1),...
    'Visible', 'off',...
    'Color', [1 0.5 0],...

```



```

        'Marker', 'o',...
        'LineStyle', 'none',...
        'DisplayName', 'Baseline Points');
% baseline
h.line(6) = line(...
    'Parent', h.ax(1),...
    'Visible', 'off',...
    'Color', [0 1 0],...
    'LineStyle', '—',...
    'DisplayName', 'Baseline');
% baseline corrected line
h.line(7) = line(...
    'Parent', h.ax(1),...
    'Visible', 'off',...
    'Color', [0 0 1],...
    'DisplayName', 'Baseline Corrected');
colors = {[1 0 0], [1 0.5 0], [1 0 1], [0 0.3 0.6], [0 0.3 0]};
% peak lines
for i = 1:5
    h.line(9+i) = line(...
        'Parent', h.ax(1),...
        'Visible', 'off',...
        'Color', colors{i},...
        'LineStyle', '—',...
        'DisplayName', sprintf('Peak %i', i));
end
% fit line
h.line(8) = line(...
    'Parent', h.ax(1),...
    'Visible', 'off',...
    'Color', [0 1 0],...
    'LineStyle', '—',...
    'DisplayName', 'Fit');

% find axes position
axpos = get(h.ax(1), 'Position');

%% Smoothing error display
% smoothing error axes
h.ax(2) = axes(...
    'Parent', h.fig,...
    'Units', 'normalized',...
    'OuterPosition', [0.35 0.24 0.65 0.19],...
    'XGrid', 'on',...
    'YGrid', 'on',...
    'YLim', [-10, 10]);
h.ax(2).YLabel.String = 'Relative Error';
h.ax(2).Title.String = 'Smoothing Error';
% smoothing error line
h.line(4) = line(...
    'Parent', h.ax(2),...
    'Visible', 'off',...
    'Color', [1 0 0],...
    'DisplayName', 'Smoothing Error');
% smoothing error text
h.text(1) = text(...
    'Parent', h.ax(2),...
    'Units', 'normalized',...
    'Position', [0.1 0.75],...
    'Visible', 'off');

%% Fitting error display
% fit error axes
h.ax(3) = axes(...
    'Parent', h.fig,...
    'Units', 'normalized',...
    'OuterPosition', [0.35 0.05 0.65 0.19],...
    'XGrid', 'on',...

```

```

        'YGrid', 'on',...
        'YLim', [-10, 10]);
h.ax(3).YLabel.String = 'Relative Error';
h.ax(3).Title.String = 'Fitting Error';
% fit error line
h.line(9) = line(...
    'Parent', h.ax(3),...
    'Visible', 'off',...
    'Color', [1 0 0],...
    'DisplayName', 'Fitting Error',...
    'Visible', 'off');
% fit error text
h.text(2) = text(...
    'Parent', h.ax(3),...
    'Units', 'normalized',...
    'Position', [0.1 0.75],...
    'Visible', 'off');
%% Common axis label display
fontsize = get(h.ax(1).YLabel, 'FontSize');
% xlabel axes
h.ax(4) = axes(...
    'Parent', h.fig,...
    'Units', 'normalized',...
    'OuterPosition', [axpos(1) 0 axpos(3) 0.05],...
    'Visible', 'off');
% xlabel text
h.xlabel = text(...
    'Parent', h.ax(4),...
    'Units', 'normalized',...
    'Position', [0.5 0.5],...
    'HorizontalAlignment', 'center',...
    'VerticalAlignment', 'middle',...
    'String', 'Raman Shift [cm-1]',...
    'FontSize', fontsize);

%% Global control panel
h.panel = uipanel(...
    'Parent', h.fig,...
    'Units', 'normalized',...
    'Position', [0 0 0.35 1],...
    'Title', 'Controls',...
    'TitlePosition', 'centertop',...
    'BorderType', 'none');

%% File Selection
% control panel
h.file.panel = uipanel(...
    'Parent', h.panel,...
    'Units', 'normalized',...
    'Position', [0 0.9 1 0.1],...
    'Title', 'File Selection');

% enable editing/apply changes
h.file.control(1) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.file.panel,...
    'Units', 'normalized',...
    'Position', [0.83 0.3 0.14 0.4],...
    'String', 'Apply',...
    'TooltipString', 'Fit spectrum',...
    'UserData', 'file',...
    'Callback', @apply);

% dropdown menu
h.file.control(2) = uicontrol(...
    'Style', 'popupmenu',...
    'Parent', h.file.panel,...
    'Units', 'normalized',...

```

```

        'Position', [0.05 0.4 0.5 0.2],...
        'String', {data(:).File},...
        'TooltipString', 'Select a spectrum',...
        'CreateFcn', @fileChange,...
        'Callback', @fileChange);

% display info
h.file.control(3) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.file.panel,...
    'Units', 'normalized',...
    'Position', [0.6 0.3 0.2 0.4],...
    'String', 'File Information',...
    'TooltipString', 'Display information about selected spectrum',...
    'Callback', @fileInfo);

%% Window Limits
% global panel
h.limits.panel(1) = uipanel(...
    'Parent', h.panel,...
    'Units', 'normalized',...
    'Position', [0 0.8 1 0.1],...
    'Title', 'Limit Selection');
% lower panel
h.limits.panel(2) = uipanel(...
    'Parent', h.limits.panel(1),...
    'Units', 'normalized',...
    'Position', [0 0 0.4 1],...
    'Title', 'Lower Limit',...
    'TitlePosition', 'centertop',...
    'BorderType', 'none');
% upper panel
h.limits.panel(3) = uipanel(...
    'Parent', h.limits.panel(1),...
    'Units', 'normalized',...
    'Position', [0.4 0 0.4 1],...
    'Title', 'Upper Limit',...
    'TitlePosition', 'centertop',...
    'BorderType', 'none');

% enable editing/apply changes
h.limits.control(1) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.limits.panel(1),...
    'Units', 'normalized',...
    'Position', [0.83 0.24 0.14 0.4],...
    'String', 'Edit',...
    'TooltipString', 'Edit limits',...
    'UserData', 'limits',...
    'Callback', @edit);
% lower limit
h.limits.control(2) = uicontrol(...
    'Style', 'edit',...
    'Parent', h.limits.panel(2),...
    'Units', 'normalized',...
    'Position', [0.2 0.3 0.6 0.4],...
    'TooltipString', 'Enter lower limit',...
    'Callback', @llimChange);
% upper limit
h.limits.control(3) = uicontrol(...
    'Style', 'edit',...
    'Parent', h.limits.panel(3),...
    'Units', 'normalized',...
    'Position', [0.2 0.3 0.6 0.4],...
    'TooltipString', 'Enter upper limit',...
    'Callback', @ulimChange);

%% Smoothing

```

```

% control panel
h.smoothing.panel = uipanel(...
    'Parent', h.panel,...
    'Units', 'normalized',...
    'Position', [0 0.7 1 0.1],...
    'Title', 'Smoothing Coefficient');

% enable editing/apply changes
h.smoothing.control(1) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.smoothing.panel,...
    'Units', 'normalized',...
    'Position', [0.83 0.3 0.14 0.4],...
    'String', 'Edit',...
    'TooltipString', 'Edit smoothing',...
    'UserData', 'smoothing',...
    'Callback', @edit);

% slider
h.smoothing.control(2) = uicontrol(...
    'Style', 'slider',...
    'Parent', h.smoothing.panel,...
    'Units', 'normalized',...
    'Position', [0.05 0.38 0.5 0.25],...
    'Max', 101,...
    'Min', 1,...
    'SliderStep', [0.02 0.10],...
    'Value', 51,...
    'BackgroundColor', [1 1 1],...
    'TooltipString', 'Adjust smoothing factor',...
    'Callback', @resmooth);

% display numerical value
h.smoothing.control(3) = uicontrol(...
    'Style', 'text',...
    'Parent', h.smoothing.panel,...
    'Units', 'normalized',...
    'Position', [0.6 0.3 0.2 0.3],...
    'String', ['Current Value: ' num2str(h.smoothing.control(2).Value)],...
    'TooltipString', 'Current smoothing factor',...
    'UserData', h.smoothing.control(2).Value);

%% Baseline
% control panel
h.baseline.panel = uipanel(...
    'Parent', h.panel,...
    'Units', 'normalized',...
    'Position', [0 0.6 1 0.1],...
    'Title', 'Baseline Editing');

% enable editing/apply changes
h.baseline.control(1) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.baseline.panel,...
    'Units', 'normalized',...
    'Position', [0.83 0.3 0.14 0.4],...
    'String', 'Edit',...
    'TooltipString', 'Edit baseline',...
    'UserData', 'baseline',...
    'Callback', @edit);

% reset baseline
h.baseline.control(2) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.baseline.panel,...
    'Units', 'normalized',...
    'Position', [0.025 0.3 0.15 0.4],...
    'String', 'Reset',...

```

```

        'TooltipString', 'Reset the baseline points to just the endpoints',...
        'Callback', @resetBaseline);

% replace removed point
h.baseline.control(3) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.baseline.panel,...
    'Units', 'normalized',...
    'Position', [0.225 0.3 0.15 0.4],...
    'String', 'Replace',...
    'TooltipString', 'Replace last removed point',...
    'Callback', @replace);

% text baseline point entry
h.baseline.control(4) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.baseline.panel,...
    'Units', 'normalized',...
    'Position', [0.425 0.3 0.15 0.4],...
    'String', 'Keyboard',...
    'TooltipString', 'Create a fit point using text entry',...
    'Callback', @allowManual);

% force point off line
h.baseline.control(5) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.baseline.panel,...
    'Units', 'normalized',...
    'Position', [0.625 0.3 0.15 0.4],...
    'String', 'Force',...
    'TooltipString', 'Create a fit point off the data',...
    'Callback', @toForce);

%% Gaussian fit controls
% control panel
h.gaussian.panel(1) = uipanel(...
    'Parent', h.panel,...
    'Units', 'normalized',...
    'Position', [0 0 1 0.6],...
    'Title', 'Fitting Controls');

% panel for changing number of fits/interacting
h.gaussian.panel(2) = uipanel(...
    'Parent', h.gaussian.panel(1),...
    'Units', 'normalized',...
    'Position', [0 0.84 1 0.16],...
    'BorderType', 'none');

% enable editing/make changes
h.gaussian.control(1) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.gaussian.panel(2),...
    'Units', 'normalized',...
    'Position', [0.83 0.3 0.14 0.4],...
    'String', 'Edit',...
    'TooltipString', 'Edit fit parameters',...
    'UserData', 'gaussian',...
    'Callback', @edit);

% run fit program
h.gaussian.control(2) = uicontrol(...
    'Style', 'pushbutton',...
    'Parent', h.gaussian.panel(2),...
    'Units', 'normalized',...
    'Position', [0.6 0.3 0.2 0.4],...
    'String', 'Fit',...
    'TooltipString', 'Run fitting algorithm',...
    'UserData', 1,...

```

```

        'Callback', @fitCurves);

% pick number of curves
h.gaussian.control(3) = uibuttongroup(...
    'Parent', h.gaussian.panel(2),...
    'Units', 'normalized',...
    'Position', [0.05 0.12 0.5 0.8],...
    'Title', 'Number of curves',...
    'BorderType', 'none',...
    'SelectionChangedFcn', @changeNum);

for i = 1:5
    % radiobox
    uicontrol(...
        'Parent', h.gaussian.control(3),...
        'Style', 'radiobutton',...
        'Units', 'normalized',...
        'Position', [0.05+0.2*(i-1) 0.37 0.1 0.4],...
        'String', num2str(i));

    % curve panel
    h.gaussian.panel(i+2) = uipanel(...
        'Parent', h.gaussian.panel(1),...
        'Units', 'normalized',...
        'Position', [0 0.84-0.168*i 1 0.168],...
        'Title', sprintf('Curve %i', i),...
        'TitlePosition', 'centertop',...
        'BorderType', 'none');

    % mean controls
    h.gaussian.fitting.mean(i) = uipanel(...
        'Parent', h.gaussian.panel(i+2),...
        'Units', 'normalized',...
        'Position', [0 0.1 0.5 0.9],...
        'Title', 'Centroid',...
        'TitlePosition', 'centertop');
    h.gaussian.fitting.lower(2*i-1) = uicontrol(...
        'Parent', h.gaussian.fitting.mean(i),...
        'Units', 'normalized',...
        'Position', [0.06 0.35 0.22 0.3],...
        'Style', 'edit',...
        'TooltipString', 'Mean lower bound (''-inf'' = no bound)',...
        'UserData', {2*i-1, 'mean'},...
        'Callback', @lbChange);
    h.gaussian.fitting.estimate(2*i-1) = uicontrol(...
        'Parent', h.gaussian.fitting.mean(i),...
        'Units', 'normalized',...
        'Position', [0.39 0.35 0.22 0.3],...
        'Style', 'edit',...
        'TooltipString', sprintf(['Mean estimate.\nIf it exceeds the ',...
        'bounds the fit algorithm defaults to the nearest bound.']),...
        'UserData', 2*i-1,...
        'Callback', @estChange);
    h.gaussian.fitting.upper(2*i-1) = uicontrol(...
        'Parent', h.gaussian.fitting.mean(i),...
        'Units', 'normalized',...
        'Position', [0.72 0.35 0.22 0.3],...
        'Style', 'edit',...
        'TooltipString', 'Mean upper bound (''inf'' = no bound)',...
        'UserData', 2*i-1,...
        'Callback', @ubChange);

    % standard deviation controls
    h.gaussian.fitting.deviation(i) = uipanel(...
        'Parent', h.gaussian.panel(i+2),...
        'Units', 'normalized',...
        'Position', [0.5 0.1 0.5 0.9],...
        'Title', 'Standard Deviation',...

```



```

        'TitlePosition', 'centertop');
h.gaussian.fitting.lower(2*i) = uicontrol(...
    'Parent', h.gaussian.fitting.deviation(i),...
    'Units', 'normalized',...
    'Position', [0.06 0.35 0.22 0.3],...
    'Style', 'edit',...
    'TooltipString', 'Standard deviation lower bound (min = 0)',...
    'UserData', {2*i, 'std'},...
    'Callback', @lbChange);
h.gaussian.fitting.estimate(2*i) = uicontrol(...
    'Parent', h.gaussian.fitting.deviation(i),...
    'Units', 'normalized',...
    'Position', [0.39 0.35 0.22 0.3],...
    'Style', 'edit',...
    'TooltipString', sprintf(['Standard Deviation estimate.',...
    '\nIf it exceeds the bounds the fit algorithm defaults ',...
    'to the nearest bound.']),...
    'UserData', 2*i,...
    'Callback', @estChange);
h.gaussian.fitting.upper(2*i) = uicontrol(...
    'Parent', h.gaussian.fitting.deviation(i),...
    'Units', 'normalized',...
    'Position', [0.72 0.35 0.22 0.3],...
    'Style', 'edit',...
    'TooltipString', 'Standard deviation upper bound (''inf'' = no bound)',...
    'UserData', 2*i,...
    'Callback', @ubChange);
end

% Disable invalid panel controls
set(findall(h.limits.panel, '-property', 'Enable'), 'Enable', 'off')
set(findall(h.smoothing.panel, '-property', 'Enable'), 'Enable', 'off')
set(findall(h.baseline.panel, '-property', 'Enable'), 'Enable', 'off')
set(findall(h.gaussian.panel, '-property', 'Enable'), 'Enable', 'off')
set(h.ax(1).Children, 'PickableParts', 'none')

% initialize and display figure
apply(h.file.control(1), [])
h.file.control(1).String = 'Apply';
h.file.control(1).Callback = @apply;
h.file.control(1).TooltipString = 'Select file and continue';
set(findall(h.file.panel, '-property', 'Enable'), 'Enable', 'on')
h.limits.control(1).Enable = 'off';
h.fig.Visible = 'on';
waitfor(h.fig)

%% Nested functions

function normed = normalize(ydata)
    % Takes a vector and linearly scales the data so the lowest value
    % is 0 and the highest value is 100
    ymin = min(ydata);
    ymax = max(ydata);
    yrng = ymax - ymin;
    normed = (ydata - ymin) / yrng * 100;
end

function updateDisp()
    % update the display information

    % reload the x and y data for the lines
    h.line(1).XData = gui.shift.orig;
    set(h.line([2:4,6:end]), 'XData', gui.shift.clip);
    h.line(5).XData = gui.bslin.pnts(:, 1);
    h.line(1).YData = gui.count.orig;
    h.line(2).YData = gui.count.clip;
    h.line(3).YData = gui.count.smth;
    h.line(4).YData = gui.error.smth;

```

```

h.line(5).YData = gui.bslin.pnts(:, 2);
h.line(6).YData = gui.bslin.base;
h.line(7).YData = gui.count.base;
h.line(8).YData = gui.count.fitd;
h.line(9).YData = gui.error.fitd;
[h.line(10:14).YData] = deal(gui.frslt.crvs{:});
% reload the string information
h.text(1).String = sprintf(['Mean: %.2f%c\n',...
    'RMS: %.2f%c\nRange: %.2f%c'],...
    mean(gui.error.smth), char(37),...
    std(gui.error.smth), char(37),...
    max(gui.error.smth) - min(gui.error.smth), char(37));
h.text(2).String = sprintf(['Mean: %.2f%c\n',...
    'RMS: %.2f%c\nRange: %.2f%c'],...
    mean(gui.error.fitd), char(37),...
    std(gui.error.fitd), char(37),...
    max(gui.error.fitd) - min(gui.error.fitd), char(37));
% recreate the legend
h.legend = legend(h.ax(1), findall(h.line([1:3, 5:8, 10:14])),...
    'Visible', 'on'));
% reset the limits
set(h.ax, 'XLim', [gui.limits.lowr, gui.limits.uppr])
end

function resetFitControls()
% reset the fit controls so that the lower limits for the means are
% -inf and for the standard deviations are 0, the estimates for the
% means are the center of the interval and the standard deviations
% are 100, and the upper limits for everything are inf
middle = (gui.shift.clip(1) + gui.shift.clip(end))/2;
set(h.gaussian.fitting.estimate(1:2:9), 'String', num2str(middle))
set(h.gaussian.fitting.estimate(2:2:10), 'String', '100')
set([h.gaussian.fitting.upper], 'String', 'inf')
set(h.gaussian.fitting.lower(1:2:9), 'String', '-inf')
set(h.gaussian.fitting.lower(2:2:10), 'String', '0')
end

function reloadFitParams()
% reload the tracking for the fit parameters using the currently
% displayed values on the uicontrols
gui.fbnds.lowr = transpose(str2num(char(h.gaussian.fitting.lower.String))); %#ok<*ST2NM>
gui.fbnds.estm = transpose(str2num(char(h.gaussian.fitting.estimate.String)));
gui.fbnds.uppr = transpose(str2num(char(h.gaussian.fitting.upper.String)));
end

function reloadBaseline()
% recalculate the baseline using the fit points and subtract it
% from the smoothed data
gui.bslin.base = pchip(gui.bslin.pnts(:, 1),...
    gui.bslin.pnts(:, 2), gui.shift.clip);
gui.count.base = gui.count.smth - gui.bslin.base;
end

function reloadSmoothing()
% reload the smoothing using the stored smoothing radius, and
% calculate the error function
gui.count.smth = smooth(gui.count.clip, gui.smthr);
gui.error.smth = (gui.count.clip - gui.count.smth);
end

function reloadLimits()
% reload the data that falls inside the stored upper and lower
% limits
J = find(gui.limits.lowr < gui.shift.orig & gui.shift.orig < gui.limits.uppr);
gui.shift.clip = gui.shift.orig(J);
gui.count.clip = normalize(gui.count.orig(J));
end

```

```

function displayFitInfo(num)
    % display the fit information in a new window

    % aggregate the information
    peaks = 1:num;
    mus = gui.frslt.nlpr(1:2:2*num-1);
    sigs = gui.frslt.nlpr(2:2:2*num);
    fwhm = 2*sqrt(2*log(2))*sigs;
    amps = gui.frslt.lpar(1:num)';
    info = [peaks; amps; mus; sigs; fwhm];
    infoString = sprintf('%i      %5.2f      %6.1f      %5.1f      %5.1f\n\n', info);
    % fit info display with save button
    hfig = figure(...
        'MenuBar', 'none',...
        'Name', 'Fit Display',...
        'NumberTitle', 'off',...
        'ToolBar', 'none',...
        'Units', 'characters',...
        'Position', [20 40 75 15]);
    display.panel = uipanel(...
        'Parent', hfig,...
        'BorderType', 'none',...
        'Units', 'normalized',...
        'Position', [0 0.15 1 0.85],...
        'Title', 'Fit Information',...
        'TitlePosition', 'centertop');
    display.save = uicontrol(...
        'Parent', hfig,...
        'Units', 'characters',...
        'Position', [20 0.75 30 2],...
        'String', 'Save Information',...
        'TooltipString', ['Save all three axes, the filename, ',...
        'the limits, smoothing factor, the baseline points, the ',...
        'fit parameters, and the fit results to a .png'],...
        'Callback', @saveInfo);
    display.header = uicontrol(...
        'Parent', display.panel,...
        'Units', 'normalized',...
        'Position', [0 0.9 1 0.1],...
        'Style', 'text',...
        'FontName', 'FixedWidth',...
        'HorizontalAlignment', 'left',...
        'String', sprintf('Peak      Amplitude      Mean      Standard Deviation      FWHM'));
    display.fit = uicontrol(...
        'Parent', display.panel,...
        'Units', 'normalized',...
        'Position', [0 0 1 0.9],...
        'Style', 'text',...
        'HorizontalAlignment', 'left',...
        'FontName', 'FixedWidth',...
        'String', infoString);
end

%% Callbacks

function edit(cbo, ~)
    % callback to execute when any 'edit' button is pressed

    % hide all curves and text, enable the appropriate controls, change
    % the edit button to an apply button
    set([h.line, h.text], 'Visible', 'off')
    set(findall(h.panel, '-property', 'Enable'), 'Enable', 'off')
    set(findall(cbo.Parent, '-property', 'Enable'), 'Enable', 'on')
    h.file.control(3).Enable = 'on';
    cbo.String = 'Apply';
    cbo.Callback = @apply;
    % evaluate based on which controls are now active
    switch cbo.UserData

```

```

case 'file'
    cbo.TooltipString = 'Select file and continue';
    h.line(1).Visible = 'on';
    gui.limits.lowr = gui.shift.orig(1);
    gui.limits.uppr = gui.shift.orig(end);
case 'limits'
    cbo.TooltipString = 'Apply limits and continue';
    h.line(2).Visible = 'on';
case 'smoothing'
    cbo.TooltipString = 'Apply smoothing and continue';
    h.line(3).Color = [1 0 0];
    h.line(3).LineStyle = '—';
    set(h.line(2:4), 'Visible', 'on')
    h.text(1).Visible = 'on';
case 'baseline'
    cbo.TooltipString = 'Apply baseline and continue';
    set(h.line(3:6), 'Visible', 'on');
    h.text(1).Visible = 'on';
    h.ax(1).ButtonDownFcn = @click;
    h.fig.UserData = 'iBeam';
    h.fig.WindowButtonMotionFcn = @setPointer;
case 'gaussian'
    cbo.TooltipString = 'Leave fitting menu';
    num = h.gaussian.control(2).UserData;
    set(findall(h.gaussian.panel(2:num+2), '-property',...
        'Enable'), 'Enable', 'on')
    set(h.line([4, 7]), 'Visible', 'on');
    h.text(1).Visible = 'on';
end
% reenable the relevant display information
updateDisp
end

function apply(cbo, ~)
    % callback to execute when any 'apply' button is pressed

    % hide all curves and text, enable the relevant controls, and
    % change the button back to an 'edit' button
    set([h.line, h.text], 'Visible', 'off')
    set(findall(h.panel, '-property', 'Enable'), 'Enable', 'off')
    cbo.String = 'Edit';
    cbo.Callback = @edit;
    set(findall(h.panel, 'String', 'Edit'), 'Enable', 'on')
    h.file.control(3).Enable = 'on';
    % apply whatever settings were just changed
    switch cbo.UserData
        case 'file'
            cbo.TooltipString = 'Edit file';
            h.smoothing.control(1).Enable = 'off';
            h.baseline.control(1).Enable = 'off';
            h.gaussian.control(1).Enable = 'off';
            h.line(1).Visible = 'on';
            gui.limits.lowr = gui.shift.orig(1);
            gui.limits.uppr = gui.shift.orig(end);
            h.limits.control(2).String = num2str(gui.limits.lowr);
            h.limits.control(3).String = num2str(gui.limits.uppr);
            reloadLimits
            gui.smthr = 51;
            h.smoothing.control(2).Value = gui.smthr;
            h.smoothing.control(3).String = ['Current Value: ' num2str(gui.smthr)];
            reloadSmoothing
            gui.bslin.pnts = [gui.shift.clip(1), gui.count.smth(1);...
                gui.shift.clip(end), gui.count.smth(end)];
            reloadBaseline
            resetFitControls
            reloadFitParams
        case 'limits'
            cbo.TooltipString = 'Edit limits';

```

```

        h.baseline.control(1).Enable = 'off';
        h.gaussian.control(1).Enable = 'off';
        h.line(2).Visible = 'on';
        reloadSmoothing
        gui.bslin.pnts = [gui.shift.clip(1), gui.count.smth(1);...
            gui.shift.clip(end), gui.count.smth(end)];
        reloadBaseline
        resetFitControls
        reloadFitParams
    case 'smoothing'
        cbo.TooltipString = 'Edit smoothing';
        h.gaussian.control(1).Enable = 'off';
        h.line(3).Color = [0 0 1];
        h.line(3).LineStyle = '-';
        set(h.line(3:4), 'Visible', 'on')
        h.text(1).Visible = 'on';
        gui.bslin.pnts = [gui.shift.clip(1), gui.count.smth(1);...
            gui.shift.clip(end), gui.count.smth(end)];
        reloadBaseline
    case 'baseline'
        cbo.TooltipString = 'Edit baseline';
        set(h.line([4, 7]), 'Visible', 'on');
        h.text(1).Visible = 'on';
        h.ax(1).ButtonDownFcn = '';
        h.fig.WindowButtonMotionFcn = '';
    case 'gaussian'
        cbo.TooltipString = 'Edit fit parameters';
        num = h.gaussian.control(2).UserData;
        set(h.line([4, 7:num+9]), 'Visible', 'on');
        set(h.text, 'Visible', 'on');
end
% update the display to show relevant information
updateDisp
end

function fitCurves(cbo, ~)
    % fit to the baseline corrected data and display the fits
    toDelete = findobj('Name', 'Fit Display');
    delete(toDelete)
    hwait = waitbar(0.33, 'Fitting...', 'Name', 'Calculation', 'WindowStyle', 'modal');
    reloadFitParams
    set(h.line(8:14), 'Visible', 'off')
    num = cbo.UserData;
    [gui.frslt.nlpr, gui.frslt.lpar] = fminspreas(funlist(1:num),...
        gui.fbnds.estm(1:num*2), gui.shift.clip, gui.count.base,...
        gui.fbnds.lowr(1:num*2), gui.fbnds.uppr(1:num*2), [],...
        optimset('MaxIter', 10000, 'MaxFunEvals', 10000));
    for j = 1:num
        gui.frslt.crvs(j) = {gui.frslt.lpar(j)*gaussian(gui.frslt.nlpr(2*j-1),...
            gui.frslt.nlpr(2*j), gui.shift.clip)};
    end
    waitbar(0.66, hwait, 'Plotting...')
    gui.count.fitd = sum(cell2mat(gui.frslt.crvs(1:num)), 2);
    gui.error.fitd = (gui.count.base - gui.count.fitd);
    h.text(2).Visible = 'on';
    set(h.line(8:num+9), 'Visible', 'on')
    updateDisp
    delete(hwait)
    displayFitInfo(num);
end

function changeNum(~, evtdata)
    % callback to change the number of curves to fit to the data
    num = str2double(evtdata.NewValue.String);
    set(findall(h.gaussian.panel(1), '-property', 'Enable'),...
        'Enable', 'off')
    set(findall(h.gaussian.panel(2:num+2), '-property', 'Enable'),...
        'Enable', 'on')

```

```

        h.gaussian.control(2).UserData = num;
    end

function lbChange(cbo, ~)
    % callback to change the lower bound for a fit parameter, checks
    % that the new value is valid or reverts to old value
    temp = str2double(cbo.String);
    ind = cbo.UserData{1};
    if isnan(temp)
        msgbox('Bounds must be numbers')
        cbo.String = num2str(gui.fbnds.lowr(ind));
    elseif temp > gui.fbnds.uppr(ind)
        msgbox('Lower bound must be less than upper bound')
        cbo.String = num2str(gui.fbnds.lowr(ind));
    elseif strcmp(cbo.UserData{2}, 'std') && temp < 0
        msgbox('Standard deviation lower bound cannot be less than 0')
        cbo.String = num2str(gui.fbnds.lowr(ind));
    else
        reloadFitParams
    end
end

function ubChange(cbo, ~)
    % callback to change the upper bound for a fit parameter, checks
    % that the new value is valid or reverts to old value
    temp = str2double(cbo.String);
    ind = cbo.UserData;
    if isnan(temp)
        msgbox('Bounds must be numbers')
        cbo.String = num2str(gui.fbnds.uppr(ind));
    elseif temp < gui.fbnds.lowr(ind)
        msgbox('Upper bound must be greater than lower bound')
        cbo.String = num2str(gui.fbnds.uppr(ind));
    else
        reloadFitParams
    end
end

function estChange(cbo, ~)
    % callback to change the estimate for a fit parameter, checks that
    % the new value is a number or reverts to old value
    temp = str2double(cbo.String);
    ind = cbo.UserData;
    if isnan(temp)
        msgbox('Estimates must be numbers')
        cbo.String = num2str(gui.fbnds.estm(ind));
    else
        reloadFitParams
    end
end

function click(cbo, ~)
    % callback for interactively changing baseline fit points when the
    % points should fall on the data
    lr = h.fig.SelectionType;
    pos = cbo.CurrentPoint;
    xrng = gui.limits.uppr - gui.limits.lowr;
    if strcmp(lr, 'normal')
        [~, index] = min(((gui.shift.clip-pos(1,1))/xrng).^2);
        if ~any(gui.bslin.pnts(:, 1) == gui.shift.clip(index))
            gui.bslin.pnts = sortrows([gui.bslin.pnts; gui.shift.clip(index), gui.count.smth(index)]);
        end
    elseif strcmp(lr, 'alt')
        [~, index] = min(((gui.bslin.pnts(:,1)-pos(1,1))/xrng).^2);
        if index == 1 || index == size(gui.bslin.pnts, 1)
            msgbox('Endpoints cannot be removed')
        else
            gui.bslin.rmvd = [gui.bslin.rmvd; gui.bslin.pnts(index, :)];
        end
    end
end

```

```

        gui.bslin.pnts(index, :) = [];
    end
end
reloadBaseline
updateDisp
end

function toForce(~, ~)
    % callback to enable setting a baseline fit point off the data
    set(h.baseline.control(1:4), 'Enable', 'off')
    h.fig.UserData = 'crosshair';
    h.ax(1).ButtonDownFcn = @forced;
end

function forced(cbo, ~)
    % callback to place a forced fit point off the data
    lr = h.fig.SelectionType;
    pos = cbo.CurrentPoint;
    if strcmp(lr, 'normal')
        if ~any(gui.bslin.pnts(:, 1) == pos(1, 1)) && ~any(gui.bslin.pnts(:, 2) == pos(1, 2))
            gui.bslin.pnts = sortrows([gui.bslin.pnts; pos(1, 1:2)]);
        end
    end
    reloadBaseline
    updateDisp
    h.fig.UserData = 'iBeam';
    h.ax(1).ButtonDownFcn = @click;
    set(h.baseline.control(1:4), 'Enable', 'on')
end

function setPointer(cbo, ~)
    % callback to change the pointer appearance while creating baseline
    pos = cbo.CurrentPoint;
    if pos(1) > 0.35 && pos(2) > 0.4
        if strcmp(cbo.UserData, 'iBeam')
            cbo.Pointer = 'ibeam';
        else
            cbo.Pointer = 'crosshair';
        end
    else
        cbo.Pointer = 'arrow';
    end
end

function replace(~, ~)
    % callback to replace points previously removed from the baseline
    % fit
    if size(gui.bslin.rmvd, 1) == 0;
        msgbox('No removed points to replace')
    else
        gui.bslin.pnts = sortrows([gui.bslin.pnts;...
            gui.bslin.rmvd(end, :)]);
        gui.bslin.rmvd(end, :) = [];
        reloadBaseline
        updateDisp
    end
end

function resetBaseline(~, ~)
    % callback to remove all user-placed baseline fit points
    bset = questdlg(sprintf(['Are you sure you want to reset the',...
        'baseline?\n All current points will be lost']),...
        'Reset Baseline', 'Reset', 'Cancel', 'Cancel');
    if strcmp(bset, 'Reset')
        gui.bslin.pnts = [gui.shift.clip(1), gui.count.smth(1);...
            gui.shift.clip(end), gui.count.smth(end)];
        gui.bslin.rmvd = [];
        reloadBaseline
    end
end

```

```

        updateDisp
    end
end

function resmooth(cbo, ~)
    % callback to resmooth the data based on a new smoothing radius,
    % also updates the display
    realspan = floor(cbo.Value);
    if ~mod(realspan, 2)
        realspan = realspan - 1;
    end
    cbo.Value = realspan;
    gui.smthr = realspan;
    h.smoothing.control(3).String = ['Current Value: ' num2str(realspan)];
    reloadSmoothing
    updateDisp
end

function llimChange(cbo, ~)
    % callback to change the lower window limit. Checks that the new
    % value is valid or reverts to old value. Also updates stored
    % information and the display
    temp = str2double(cbo.String);
    if isnan(temp)
        msgbox('Limits must be numbers')
    elseif temp > gui.limits.uppr
        msgbox('Lower limit must be less than upper limit')
    elseif temp < gui.shift.orig(1)
        msgbox('Lower limit cannot be less than lowest data point')
    else
        [~, ind] = min(abs(gui.shift.orig - temp));
        gui.limits.lowr = gui.shift.orig(ind);
    end
    cbo.String = num2str(gui.limits.lowr);
    reloadLimits
    updateDisp
end

function ulimChange(cbo, ~)
    % callback to change the upper window limit. Checks that the new
    % value is valid or reverts to old value. Also updates stored
    % information and the display
    temp = str2double(cbo.String);
    if isnan(temp)
        msgbox('Limits must be numbers')
    elseif temp < gui.limits.lowr
        msgbox('Upper limit must be greater than lower limit')
    elseif temp > gui.shift.orig(end)
        msgbox('Upper limit cannot be greater than highest data point')
    else
        [~, ind] = min(abs(gui.shift.orig - temp));
        gui.limits.uppr = gui.shift.orig(ind);
    end
    cbo.String = num2str(gui.limits.uppr);
    reloadLimits
    updateDisp
end

function fileChange(cbo, ~)
    % callback to change which data set is being examined
    num = cbo.Value;
    gui.shift.orig = data(num).Shift;
    gui.count.orig = normalize(data(num).Counts);
    h.line(1).XData = gui.shift.orig;
    h.line(1).YData = gui.count.orig;
end

function fileInfo(~, ~)

```



```

% callback to display the information collected about the currently
% selected file
ind = h.file.control(2).Value;
infoFig = figure(...
    'MenuBar', 'none',...
    'ToolBar', 'none',...
    'Name', 'File Information',...
    'NumberTitle', 'off',...
    'Units', 'characters',...
    'Position', [100 40 100 20],...
    'WindowStyle', 'modal');
info = {...
    sprintf('Filename: %s', data(ind).File);...
    sprintf('Dive: %s', data(ind).Dive);...
    sprintf('Spectrum Type: %s', data(ind).Type);...
    sprintf('Species: %s', data(ind).Species);...
    sprintf('Depth: %.0fm', data(ind).Depth);...
    sprintf('Temperature: %.2f %cC', data(ind).Temperature, char(176));...
    sprintf('Salinity: %.2f pss-78', data(ind).Salinity);...
    sprintf('Pressure: %.0f dbar [abs]', data(ind).Pressure);...
    sprintf('Oxygen Level: %.3f mL/L', data(ind).Oxygen);...
    sprintf('Jelly Number: %i', data(ind).Jelly);...
    sprintf('Jelly Position: %i', data(ind).Position);...
    sprintf('Exposure: %i sec x 32', data(ind).Exposure);...
    sprintf('Notes:\n%s', data(ind).Notes)};
uicontrol(...
    'Style', 'text',...
    'Parent', infoFig,...
    'Units', 'normalized',...
    'Position', [0 0 1 1],...
    'String', info,...
    'HorizontalAlignment', 'left');
waitfor(infoFig)
end

function saveInfo(~, ~)
% callback to save the displayed info to a color .ps file. Location
% is up to the user, as is the name
save.fig = figure(...
    'MenuBar', 'none',...
    'ToolBar', 'none',...
    'Position', [0 0 1920 1080],...
    'PaperOrientation', 'landscape',...
    'Visible', 'off');
save.ax(1, :) = copyobj([h.ax(1), h.legend], save.fig);
save.ax(1, 1).Title.String = 'Processed data, fit, and peaks';
save.ax(2, :) = copyobj(h.ax(2), save.fig);
save.ax(3, :) = copyobj(h.ax(3), save.fig);
save.ax(4, :) = copyobj(h.ax(4), save.fig);
ind = h.file.control(2).Value;
file = sprintf('Filename: %s', data(ind).File);
dive = sprintf('Dive: %s', data(ind).Dive);
type = sprintf('Spectrum Type: %s', data(ind).Type);
species = sprintf('Species: %s', data(ind).Species);
depth = sprintf('Depth: %.0f', data(ind).Depth);
temp = sprintf('Temperature: %.2f%cC', data(ind).Temperature, char(176));
pressure = sprintf('Pressure: %.0f dbar [abs]', data(ind).Pressure);
salinity = sprintf('Salinity: %.2f pss-78', data(ind).Salinity);
oxygen = sprintf('Oxygen Level: %.3f mL/L', data(ind).Oxygen);
jelly = sprintf('Jelly Number: %i', data(ind).Jelly);
position = sprintf('Jelly Position: %i', data(ind).Position);
exposure = sprintf('Exposure: %i sec x 32', data(ind).Exposure);
notes = sprintf('Notes:\n %s', data(ind).Notes);
llim = sprintf('Lower Limit: %.1f', gui.limits.lowr);
ulim = sprintf('Upper Limit: %.1f', gui.limits.uppr);
smoothing = sprintf('Smoothing: %i', h.smoothing.control(2).Value);
basepoints = sprintf('%.1f ', gui.bsline.pnts(:, 1));
basestring = sprintf('Baseline Shifts: [%s]', basepoints);

```

```

paramheader = [...
    'Fit Parameters:                                     ';...
    '              mean                standard deviation ';...
    'peak      lower estimate upper      lower estimate upper'];
num = h.gaussian.control(2).UserData;
ml = gui.fbnds.lowr(1:2:2*num);
me = gui.fbnds.estm(1:2:2*num);
mu = gui.fbnds.uppr(1:2:2*num);
sl = gui.fbnds.lowr(2:2:2*num);
se = gui.fbnds.estm(2:2:2*num);
su = gui.fbnds.uppr(2:2:2*num);
params = [1:num; ml; me; mu; sl; se; su];
paramstring = sprintf('%i      %4.0f      %4.0f      %4.0f      %3.0f      %3.0f      %3.0f\n', params);
% Whitespace is necessary to avoid errors and properly space
resultheader = [...
    'Fit Results:                                     ';...
    'peak amplitude mean      STD      RMS'];
mus = gui.frslt.nlpr(1:2:2*num);
sigs = gui.frslt.nlpr(2:2:2*num);
rms = 2*sqrt(2*log(2))*sigs;
amps = gui.frslt.lpar(1:num)';
results = [1:num; amps; mus; sigs; rms];
resultstring = sprintf('%i      %5.2f      %6.1f      %5.1f      %5.1f\n', results);
info = {file; dive; type; species; depth; temp; pressure;...
    oxygen; salinity; jelly; position; exposure; notes; llim;...
    ulim; smoothing; basestring; paramheader; paramstring;...
    resultheader; resultstring};
uicontrol(...
    'Style', 'text',...
    'Parent', save.fig,...
    'Units', 'normalized',...
    'Position', [0 0 0.35 1],...
    'FontName', 'FixedWidth',...
    'BackgroundColor', [1 1 1],...
    'FontSize', 6,...
    'String', info,...
    'HorizontalAlignment', 'left');
[fname, pname] = uiputfile('*.ps', 'Save Data', strrep(data(ind).File, '.asc', '.ps'));
if ~isequal(fname, 0)
    saveas(save.fig, [pname fname], 'psc')
else
    warndlg('User canceled save', 'Save Failed')
end
end

function allowManual(cbo, ~)
% callback to allow manual entry of fit point information
set(findall(cbo.Parent, '-property', 'Enable'), 'Enable', 'off')
set(cbo,...
    'Enable', 'on',...
    'Style', 'edit',...
    'String', '',...
    'TooltipString', 'Enter x-coordinate of baseline point',...
    'Callback', @manualEntry)
end

function manualEntry(cbo, ~)
% callback to manually add a fit point to the data, checks that the
% input is valid. Also updates the display
val = str2double(cbo.String);
if isnan(val)
    msgbox('x-coordinate must be a number')
else
    [~, index] = min(abs(gui.shift.clip - val));
    if ~any(gui.shift.clip(index) == gui.bslin.pnts(:, 1))
        point = [gui.shift.clip(index), gui.count.smth(index)];
        gui.bslin.pnts = sortrows([gui.bslin.pnts; point]);
        if size(gui.bslin.rmvd, 2) > 0

```

