

# FROG FastGuide Tutorial

---

© 2004 Femtosoft Technologies  
3686 Virden Ave., Oakland, CA 94619

---

# Table of Contents

Generating Traces from Theory	5
Run the Algorithm	8
Algorithm Strategies – why does the error	
jump around?	11
More Theory Pulses	12
Experimental Data	12
Reading In Electric Fields	15
The Log	15
Fixing Up a Bad Trace	17
Sampling FROG Data	24
Experimental Settings	27
Calibration Errors	27
The Marginals	32
Using Marginals in the Software	33

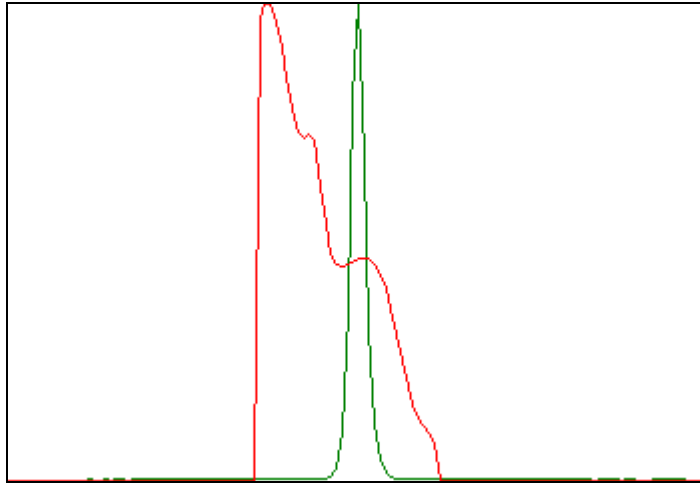
# Introduction to FROG

## *Measuring Ultrashort Pulses*

The FROG (Frequency-Resolved Optical Gating) technique is a method for measuring the complete intensity and phase of ultrashort laser pulses. The technique itself is quite interesting, and the physics are very rich. In this tutorial, we will not attempt to explain the scientific basis of FROG – that has been addressed in the literature quite adequately. Instead, we will focus on the practical aspects necessary to use the FROG software. If you are interested in the physics and mathematics of FROG, a good place to start is “Frequency-Resolved Optical Gating: The Measurement of Ultrashort Laser Pulses” by Rick Trebino.

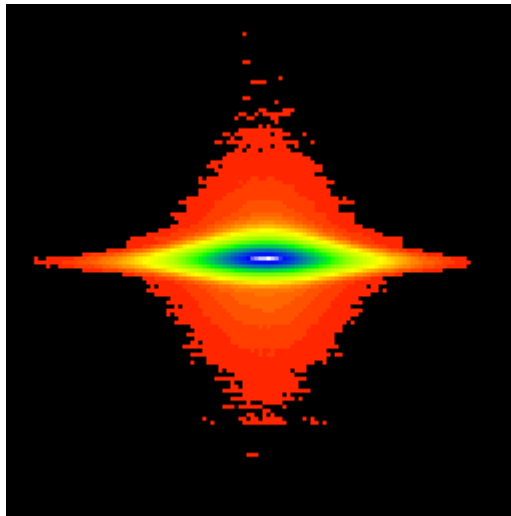
For our purposes, what is important to know is that every unique laser pulse produces a unique FROG trace. Laser pulses are characterized by the intensity and phase as a function of time of the slowly-varying envelope of the pulse (with the carrier frequency removed). Equivalently, we could use the intensity and phase as a function of frequency (or wavelength). The two representations are of course related by the Fourier Transform.

Moving from a one-dimensional representation of a laser pulse



A laser pulse represented as intensity (green) and phase (red) as a function of frequency.

to the two-dimensional FROG trace



A FROG trace.

is quite easy. However, starting from the FROG trace and inverting the transformation to get the electric field envelope is quite difficult, and in fact has no analytic solution. Instead, we rely on iterative algorithms applying arcane transformations from the areas of science like signal analysis, phase retrieval, and time-frequency analysis.

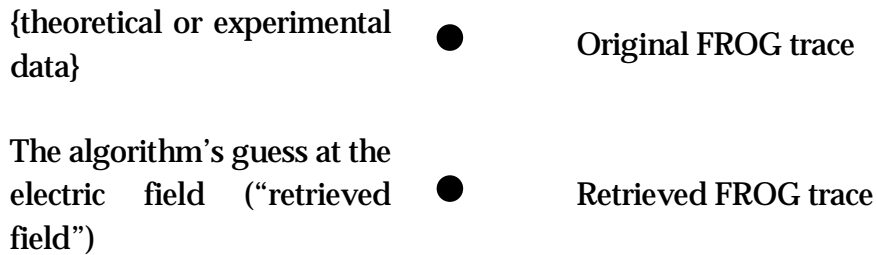
The FROG trace is an interesting representation. The x-axis corresponds to a temporal coordinate, while the y-axis corresponds to a frequency

coordinate. Normally one expects to see a signal plotted as a function of time *or* frequency, not both. To top it off, the two axes are related by the Fourier Transform relationship, thus fixing the aspect ratio of the FROG data.

The entire purpose of the FROG software package is to take a FROG trace (typically experimentally measured) and find the electric field envelope that created it.

When running the software, there are always two FROG traces: the original (sometimes called measured, experimental, or target trace) and the retrieved trace. The original trace is the FROG data that we are presented with (usually by measurement in the laboratory, but there are other ways to get FROG traces, as we'll see below). The job of the algorithm is to find an electric field that produces a FROG trace as close as possible to the original. This is an iterative process, and at any given point in time the algorithm might be more or less correct. The FROG trace that is generated from the current guess at the electric field is called the retrieved trace. The goal, restated, is to make the retrieved trace as close as possible to the original.

You might think of it this way:



The algorithm makes successive guesses at the retrieved electric field in order to minimize the difference between the original and retrieved FROG traces.

Let's see how it works.



# Your First FROG Trace

## *Create and retrieve simple FROG traces*

**T**o begin, we need a FROG trace to analyze. There are three ways to obtain a FROG trace using the FROG software. These methods are to generate a FROG trace from theoretical parameters, read in experimental FROG data, or to read in an electric field envelope.

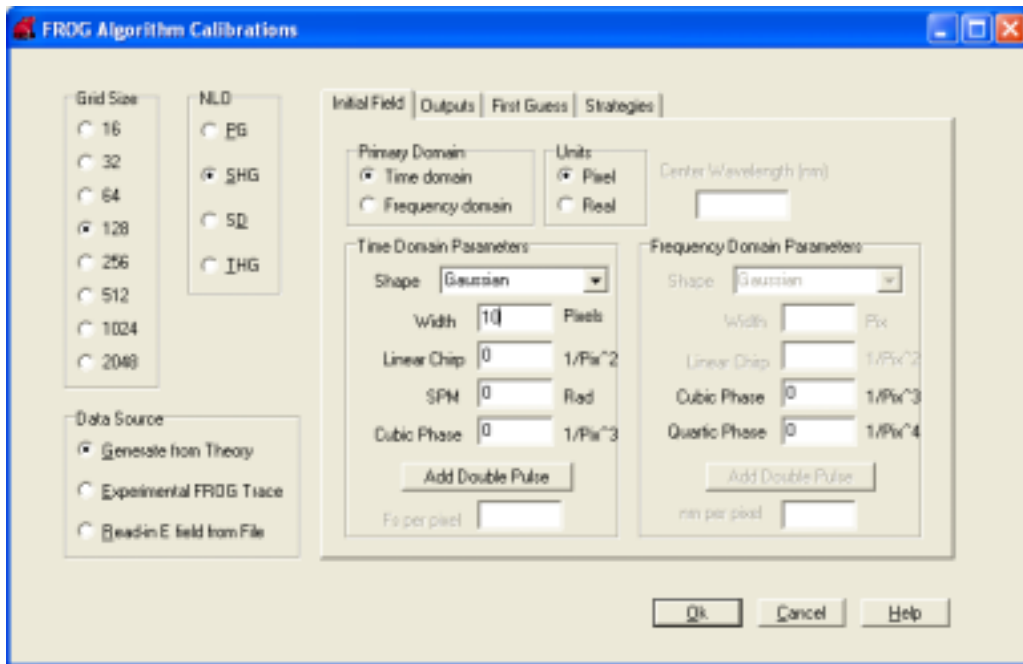
## Generating Traces from Theory

The simplest and quickest way to generate a FROG trace is to just make one up! Start the FROG program (use the Start menu or double click FROG3.exe). Begin FROG by performing one of these three actions:

- Click the “F” button in the upper left
- Use the **Run -> FROG** menu item
- Press **Ctrl+F**.

This will bring up the **FROG Algorithm Calibrations** window. If it's not already selected, select the **Generate From Theory** option in the **Data Source** box on the lower left.

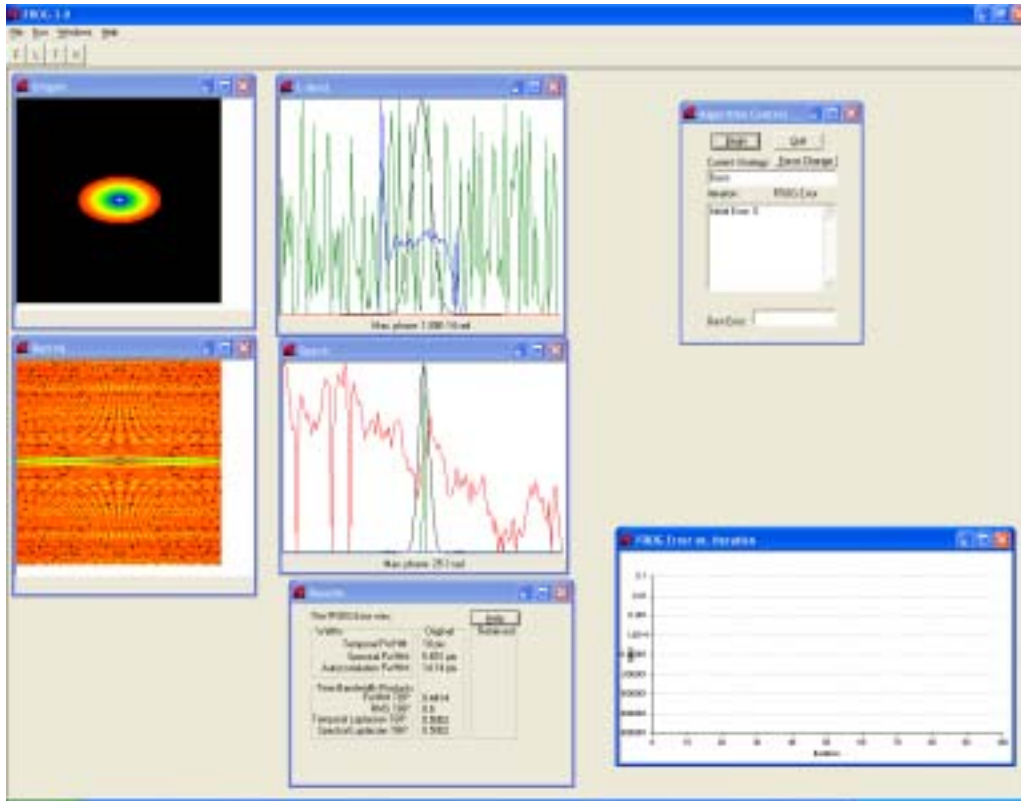




The FROG Algorithm Calibrations window.

The window appears with a set of data that represents a transform-limited pulse with a 10-pixel FWHM (full-width at half-maximum in intensity). We know the pulse is transform limited because the phase distortion boxes (Linear Chirp, SPM, etc.) are all set to zero. This simple pulse will be fine for our purposes here.

Click **Ok**.



The FROG main screen.

Now the main screen appears. Note that your windows may not appear in exactly the same location as the screen shot. You can adjust, resize and move the windows around. The program will remember your settings next time you run the software. If, for some reason, you ever want to return to the default window sizes and locations, you can do so by selecting **Windows -> Use Default Positions** from the main menu.

There's a lot going on here, so let's walk through the windows.

The upper left window is the FROG trace of the pulse we selected – in this case, a transform-limited 10-pixel-wide Gaussian pulse. As befits such a benign pulse, the FROG trace is quite docile – in fact, it is Gaussian in shape in both dimensions. This is the “target” or original FROG trace – the one that the algorithm will be trying to match.

To the right of that window is the electric field plotted as a function of time. The intensity of the original field is in black and the phase in blue. Superimposed on the same graph is the intensity (green) and phase (red) of the initial guess at the retrieved field. The default guess is random noise for the intensity and a flat phase. This seems to be a pretty effective initial guess. No one really knows why.

The plot of intensity is always normalized to unity. Phase, on the other hand, has real units (radians) and one cannot arbitrarily renormalize those units. At the bottom of the window, there's a small legend that reads something like: "Max Phase: 3.2e-14 rad". This indicates that the phase axis is normalized so that the bottom of the graph is phase = 0, and the top is phase = 3.2e-14 radians – in this case, essentially zero. Whenever you try to evaluate the phase profile of a pulse, be sure to note the "Max Phase" value, since the phase graph will always extend from the top to the bottom of the graph area. When Max Phase is below, say, about 1 radian, phase distortions are typically pretty negligible.

Below the field window is the electric field plotted as a function of frequency (or equivalently, wavelength). And on the bottom left is the retrieved FROG trace, generated from the initial guess at the retrieved electric field. At the moment it's a bit of a mess, since the initial guess at the retrieved field is random intensity noise.

On the upper right is the **Algorithm Control** window, which allows you to start and stop the retrieval, and also shows you the progress of the retrieval.

At the bottom is the **Results** window, which shows some measurements that characterize the electric fields. In this case, where we are generating a trace from theoretical parameters, we know the exact original electric field, so the original measurements are already shown on this window. If our retrieval algorithm succeeds, the retrieved field measurements should be very close to the original ones.

Finally, at the bottom right is a graph that will plot the error between the original and retrieved FROG traces as the algorithm iterates away. This gives a nice visual window into how things are going.

Run the Algorithm

Let's run the algorithm. Press the **Begin** button on the **Algorithm Control** window.

You should see the retrieved trace rapidly approach the same shape and size as the original trace. When you think they are close enough, you can press the **Stop** button on the **Algorithm Control** window.

When the algorithm stops, it writes the results to disk. We won't go into the details of the files written or their format; instead, consult the online help.

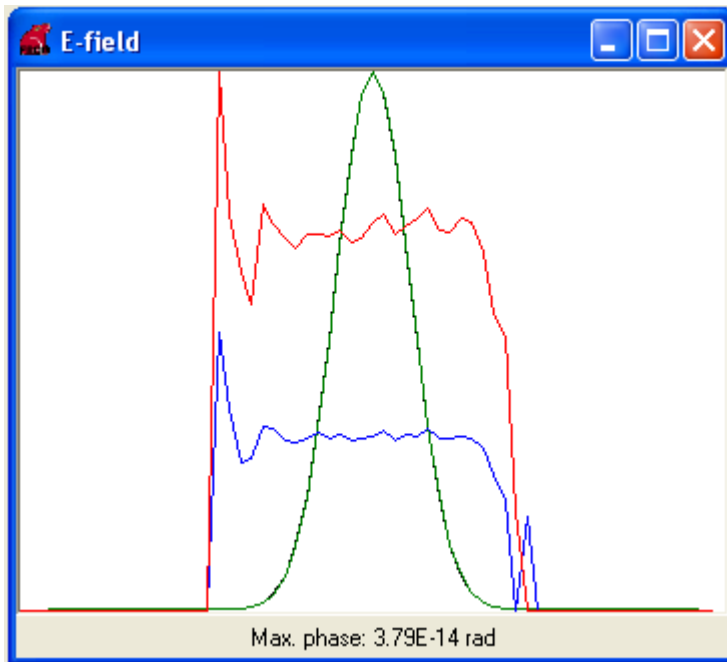
The online help is accessible by pressing F1 while running the FROG program, or by using the Help -> Contents menu item. Also, most of the windows have a help button for context sensitive help (F1 is also context sensitive).

Note that the program uses the same file names for each run (this is to make it easier to read the data into graphing programs), and thus it will overwrite previous data if it exists. You can uncheck the **Always Overwrite** checkbox on the **Outputs** tab of the **FROG Algorithm Calibrations** to change this behavior. If you are doing several runs in a row for which you want to keep the data, the suggested way of working is to put each FROG trace into a separate subdirectory. The output files will be written to the same subdirectory as the FROG trace data file, so that all the data is preserved.

Look at the **Results** window. You should see that the measured parameters (FWHM etc.) of the retrieved field are pretty close to the original field.

If you let the algorithm run long enough, you'll reach errors as low as  $1e-17$ . This indicates that we are as close as we can get to the original trace – within the roundoff errors of the computer's internal floating point number representation!

You might note that even though the convergence is superb, the two fields are not identical. For example, here's the two fields as a function of time:



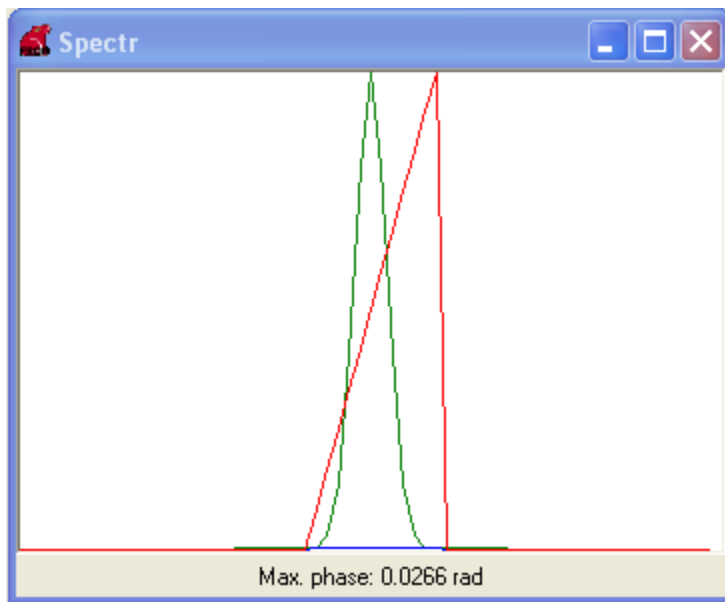
After retrieving the pulse.

Note that although the intensities look to be in good agreement, the phases (red and blue curves) are not. However, look at the phase max: it's  $3.8E-14$  radians. That means that the differences in the phase curves are only a few parts in  $10^{14}$ . That's

negligibly small, and is probably just the roundoff error in the floating point numbers.

Also, please realize that phase is not a real quantity – it's only invented for humans. The FROG algorithm deals in a complex field with real and imaginary parts. The phase is calculated by the arctan of the imaginary part divided by the real part. Let's say, for example, that the imaginary part should be zero. A small amount of numerical (or experimental!) noise can then cause the phase to flip from positive to negative. In general, in areas where the intensity is small, the phase becomes nonsensical because of noise. Therefore, on the graphs the phase is not plotted where the intensity is less than 0.0001 of the peak. This "phase blanking level" can be set from the **Outputs** tab. Just be aware that the phase is much more sensitive to noise than the intensity is.

Another interesting observation comes up from the spectrum of the retrieved field.



Spectra of the retrieved fields.

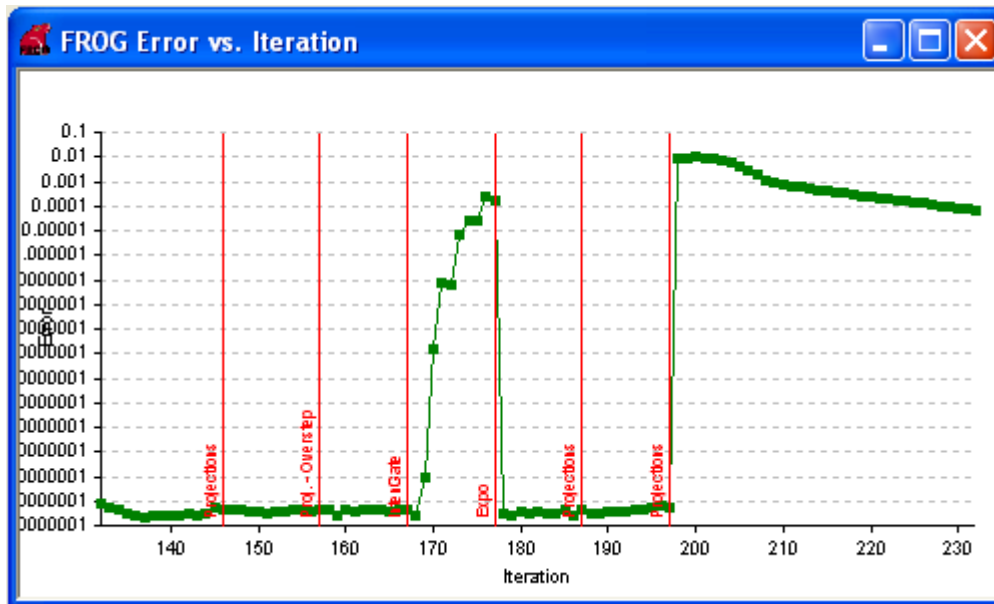
Here the agreement is so good that you can't even see the original field anymore. But notice that although the original phase (blue) is flat as a pancake, the retrieved phase (red) is linear. What does that mean?

The key fact here is that a linear phase in the frequency domain is equivalent to a temporal shift in the time domain. FROG is insensitive to the location of the temporal origin (i.e.,  $E(t)$  has the same FROG trace as  $E(t - t_0)$ ). In fact, in a free-running FROG algorithm you'll notice that the location of the pulse tends to drift in time. Therefore, the software recenters the pulse at  $t = 0$  on every iteration.

However, the recentering is not perfect, and in fact might be off by as much as half a pixel. This leads to a linear phase in the frequency domain. However, the two fields are still equivalent.

Algorithm Strategies – why does the error jump around?

If you let the algorithm run long enough, you'll start to see the FROG error actually get worse, especially when the algorithm changes “strategies”, like this:



Example of FROG error becoming unstable.

This is actually a feature, not a bug! Retrieving a FROG trace is akin to a multidimensional minimization problem. The biggest danger in doing multidimensional minimizations is in getting stuck in local minima. For an analytically generated FROG trace like the one we just used (perfect data), the danger is small. But for experimental data, with its inevitable noise and minor distortions, the probability of getting stuck increases tremendously.

In order to combat this tendency, the FROG software applies several different methods of obtaining new guesses at the retrieved field. These methods are called *strategies*. The FROG algorithm will stick with a given strategy as long as it's working. If the error stops decreasing at some minimum rate, the algorithm will declare that strategy to be in stagnation and try another one. When the strategy is switched, a red vertical line appears on the error graph, as seen above.

However, sometimes even switching strategies is not enough to jump out of a local minimum. Therefore, from time-to-time, the program will start over from noise, or simply perturb the current field, or play some other trick like that to try to jar the

algorithm out of a local minimum and into a more global minimum. It's possible to customize this behavior from the **Strategies** tab of the **FROG Algorithm Calibrations** window.

Don't worry when the error goes up: the algorithm always remembers the very best error that it has ever seen. When you stop the algorithm, it will remember the field that produced the best error and it will graph and save those values, not the values it had when you happened to push the stop button. So it's not necessary to try to push the stop button at exactly the best error!

More Theory Pulses

Feel free to play around with the "Generate From Theory" facility. It's a great way to get a feel for the algorithm, and for FROG in general. The equations governing the generated electric field are in the online help.

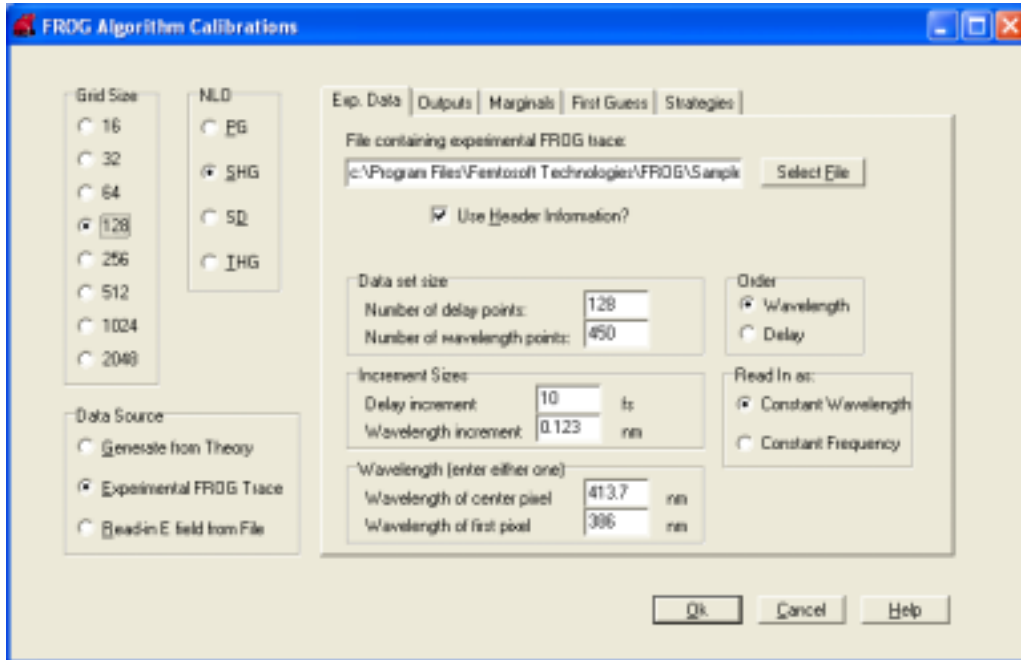
## Experimental Data

FROG traces are most commonly generated from experiments. The FROG program comes with a sample experimental data file to see how it works.

Stop the algorithm if it's running, and press Ctrl+F (or any of the other ways to start the FROG algorithm) again. In the **FROG Algorithm Calibrations** window, make these changes:

- Change the **Grid Size** to 128. The data file that we are about to use looks nicest on a grid of 128.
- Change the **NLO** to **SHG**. This setting governs the optical nonlinearity used to retrieve the pulse, and of course needs to match that of the experiment. In this case, the data was SHG FROG.
- Change the **Data Source** to **Experimental FROG Trace**. This tells the program that we are going to read in a FROG trace measured from the lab.
- Press the **Select File** button. Browse to the installation directory for FROG. In the subdirectory called **Samples** select the file named 042194.12H.
- Click the **Use Header Information** checkbox. The data calibrations are attached as headers to the actual FROG data. Checking this box means that we will use that data. Note that when you check the box, the data calibrations are filled in. You can open the data file with a text editor to see the actual data. Also, the format is documented in the online help.

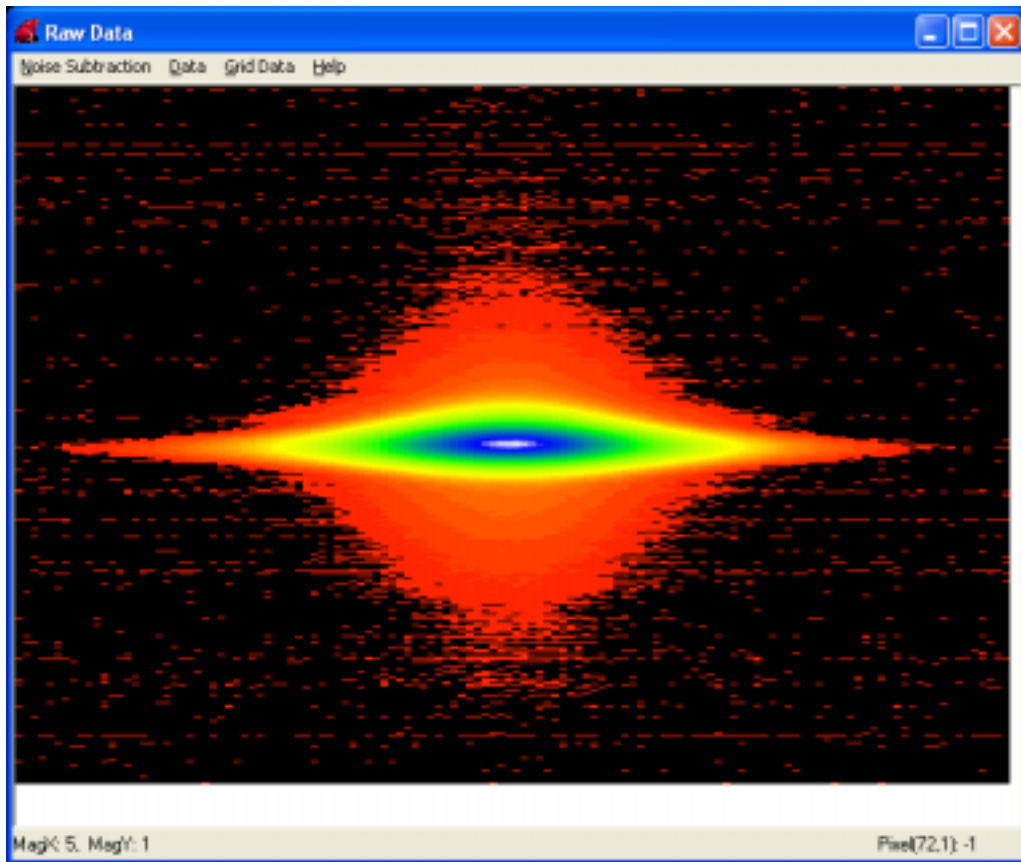
The window should look like this:



FROG Algorithm Calibrations window prepared for experimental data retrieval.

We'll come back to discuss some of these points later, but for now just click **OK**. The **Raw Data** window opens. This window is used to prepare the raw (experimental) data for use in the FROG algorithm.





The Raw Data window.

The data in this file is pretty good FROG data. However, we do notice a little bit of noise surrounding the actual trace. Noise is bad. To get rid of it, we can select the **Noise Subtraction -> Cleanup Pixels** menu item once or twice. Now the data is ready: Press the **Grid Data** menu item.

The next screen should look familiar: we're back to the main retrieval screen! Start the algorithm and see how well this data can be retrieved. You should be able to get a FROG error below 0.001. As far as I know, this trace still has the best FROG error ever measured.

In general, you should always expect errors below 0.01. For SHG, which is a background-free measurement, for grid size  $N = 128$ , errors of less than 0.005 should be readily achievable. Low error and nice visual correspondence of the original and retrieved traces indicate a good convergence.

Note that when you stop the retrieval, the **Results** window only shows the Retrieved field parameters. That's only natural – if we knew the original field parameters we wouldn't need FROG!

There's a lot more to getting good experimental FROG data. We'll address all of this in another section.

## Reading In Electric Fields

There is one last way to make a FROG trace, and that is to read in a complex electric field envelope and to analytically generate the FROG trace from that field. This would be most useful if, for instance, you are using some numerical or theoretical model to predict the outcome of an experiment. You could then read that field into the FROG program and see the FROG trace it generates. In this case, retrieving the field doesn't really serve much purpose, except to see how easy or difficult it is.

Go back to the **FROG Algorithm Calibrations** screen. Change the **Data Source** to **Read-in E Field from File**. Press the **Select File** button. Browse to the same **Samples** directory that you found the FROG data in, and select the SampleE.dat file. Again, check the **Use Header Information** checkbox. Click **OK** and run the algorithm.

This pulse is a Gaussian pulse with 4 radians of self-phase modulation.

To find out more on how to read-in fields from data files, consult the online documentation.

## The Log

After doing so many FROG retrievals, we may start to forget exactly what we did. FROG will store the results of all of your runs in a history log. You can access this by either

- Pressing Ctrl+L.
- Using the **Run -> Log** menu item.
- Pressing the little "L" button.

The log will show the details of the retrievals during the current run of the program.

	NLO	N	File	FWHM	Error
	SHG	128	None Selected	9.998 fs	6.606E-5
	PG	128	None Selected	10.03 fs	5.811E-5
	PG	128	None Selected	10 fs	2.327E-5
	SHG	128	042194.12H	137.2 fs	0.00083

Display Log

The history log.

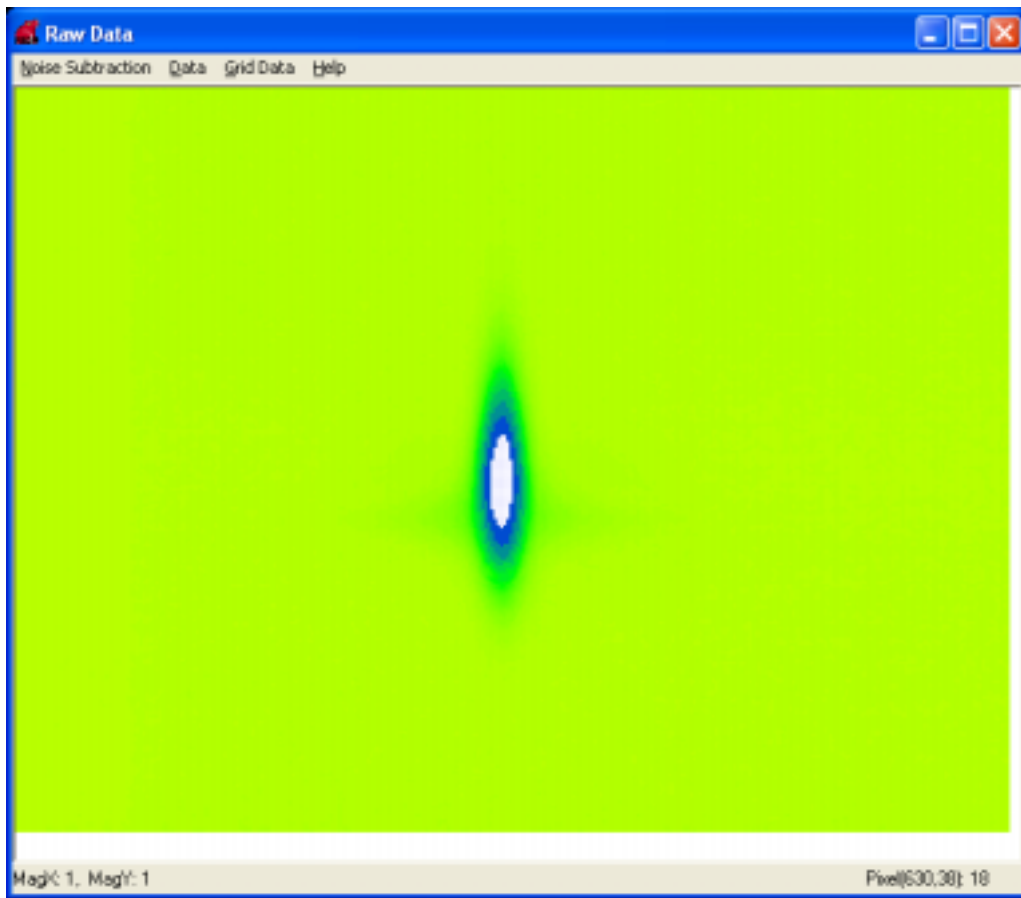
You can select any row and press **Display Log** to see the log data. This data is identical to that written to the frog.dat output file on the disk.

# Getting Good Experimental Data

## *The ins and outs of getting good retrievals*

### Fixing Up a Bad Trace

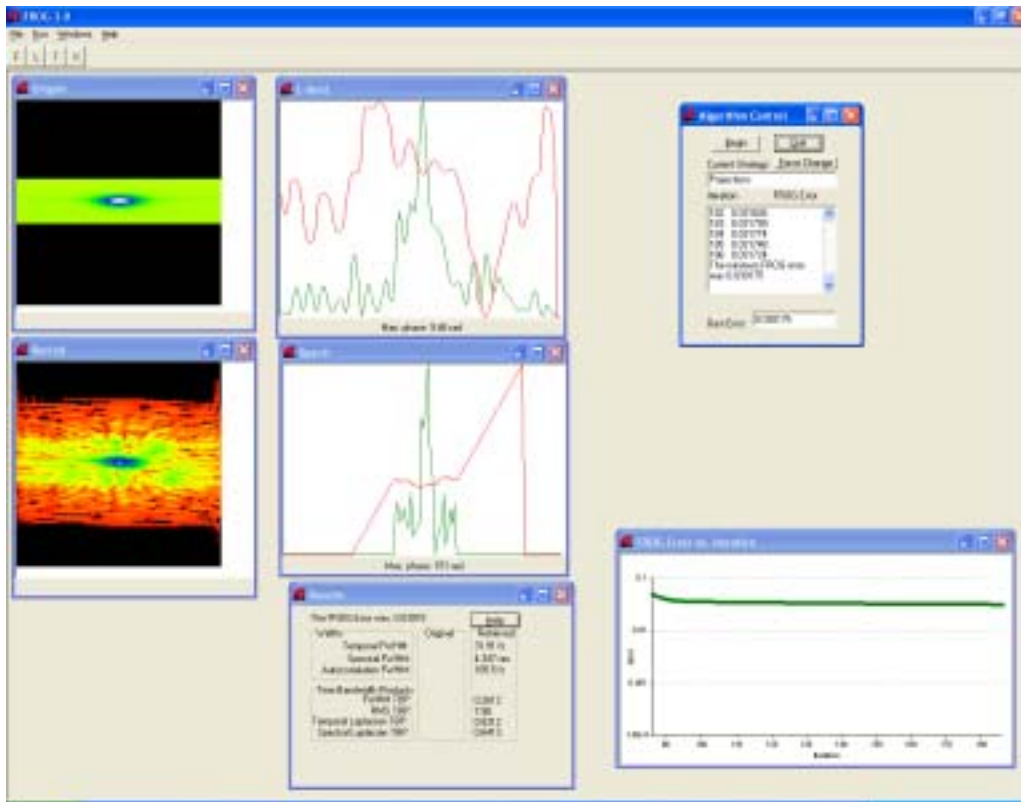
For starters, let's examine some bad data and see if we can make some sense out of it. Here's some data that I hope you will recognize as bad.



Bad data.

This data file is called NoisyData.dat in the **Samples** subdirectory. It is an SHG FROG trace, and set the grid size to 128.

You can see that there is an enormous noise floor to this trace. In fact, it's hard to see the trace for the noise! Retrieving this trace as is leads to garbage:

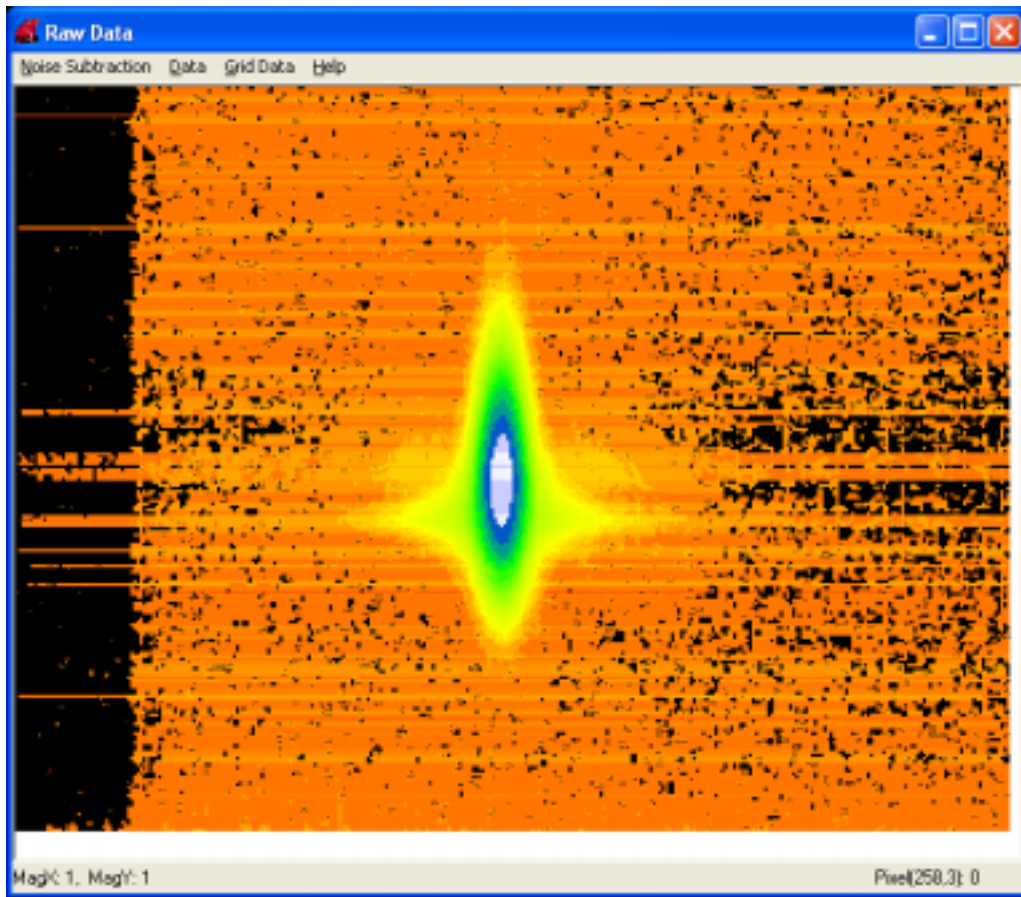


Lousy retrieval.

The noise has totally ruined the retrieval. So let's try to fix it.

The **Raw Data** window has a number of facilities for preparing data for FROG retrieval. The first is the **Noise Subtraction** menu. The first thing to try is the **Full Spectrum** noise subtraction. This mode reads the left and right edges of the data, and constructs a “background spectrum” to be subtracted.

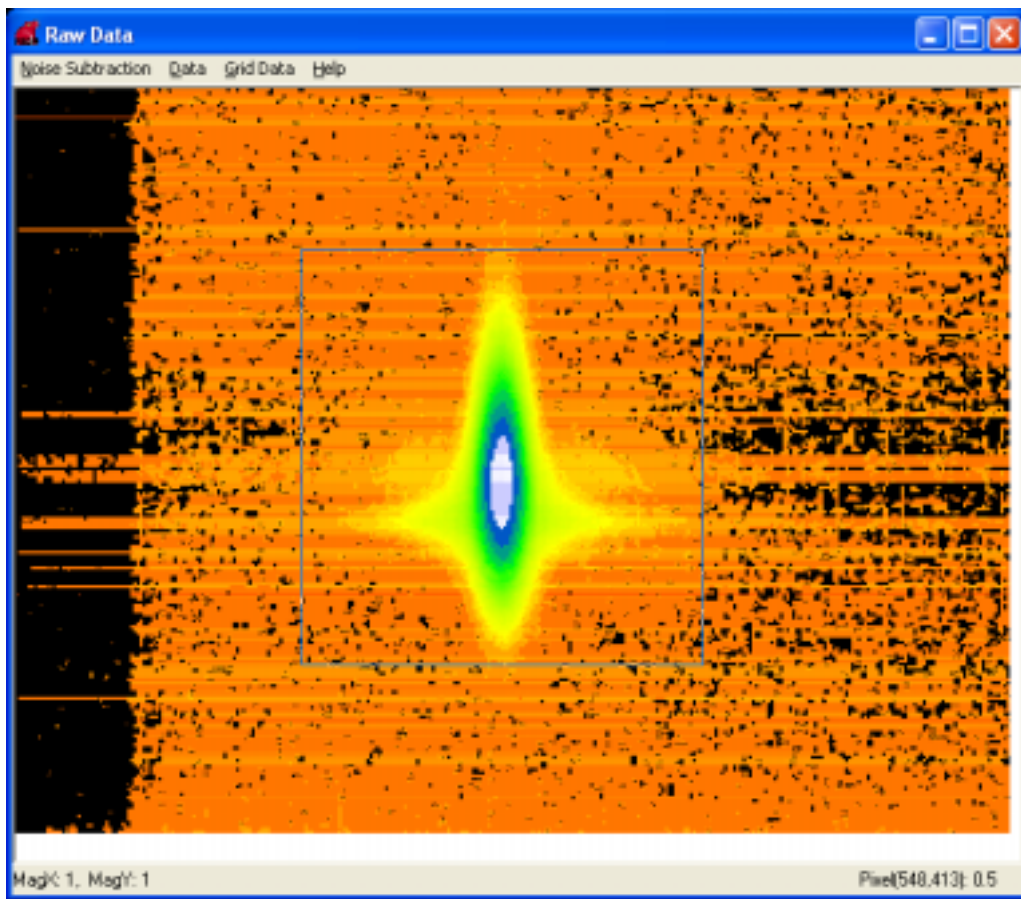
After a full spectrum subtraction we have:



After a full spectrum background subtraction.

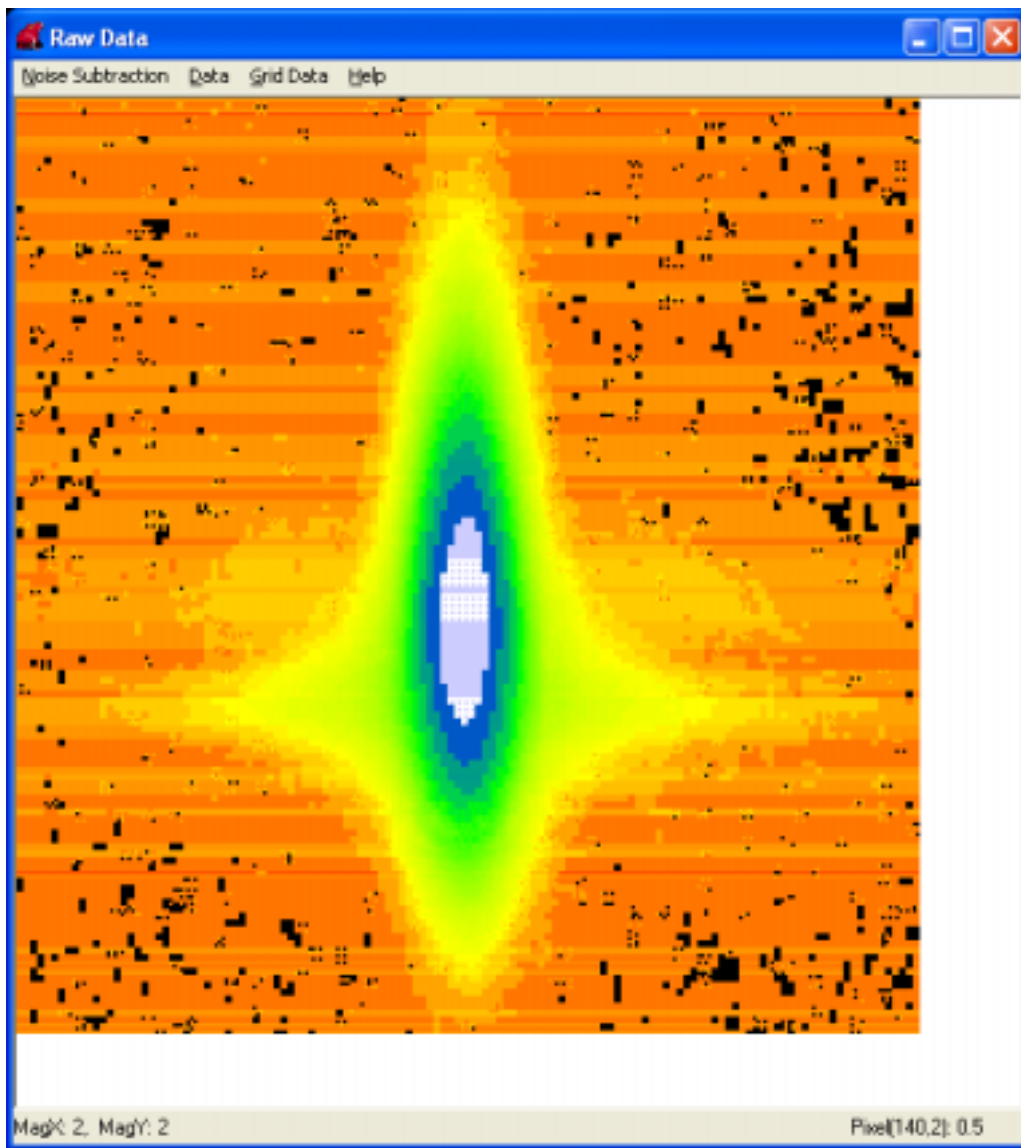
Well, that's better, but we see a new problem. The noise on the left side of the trace is less than on the right, thwarting our attempts at full spectrum subtraction. If only we could just erase some of that far-flung noise...

Enter the data extraction mode. Using the mouse, draw a rectangle around the "good" part of the data:



Drawing a data extraction rectangle.

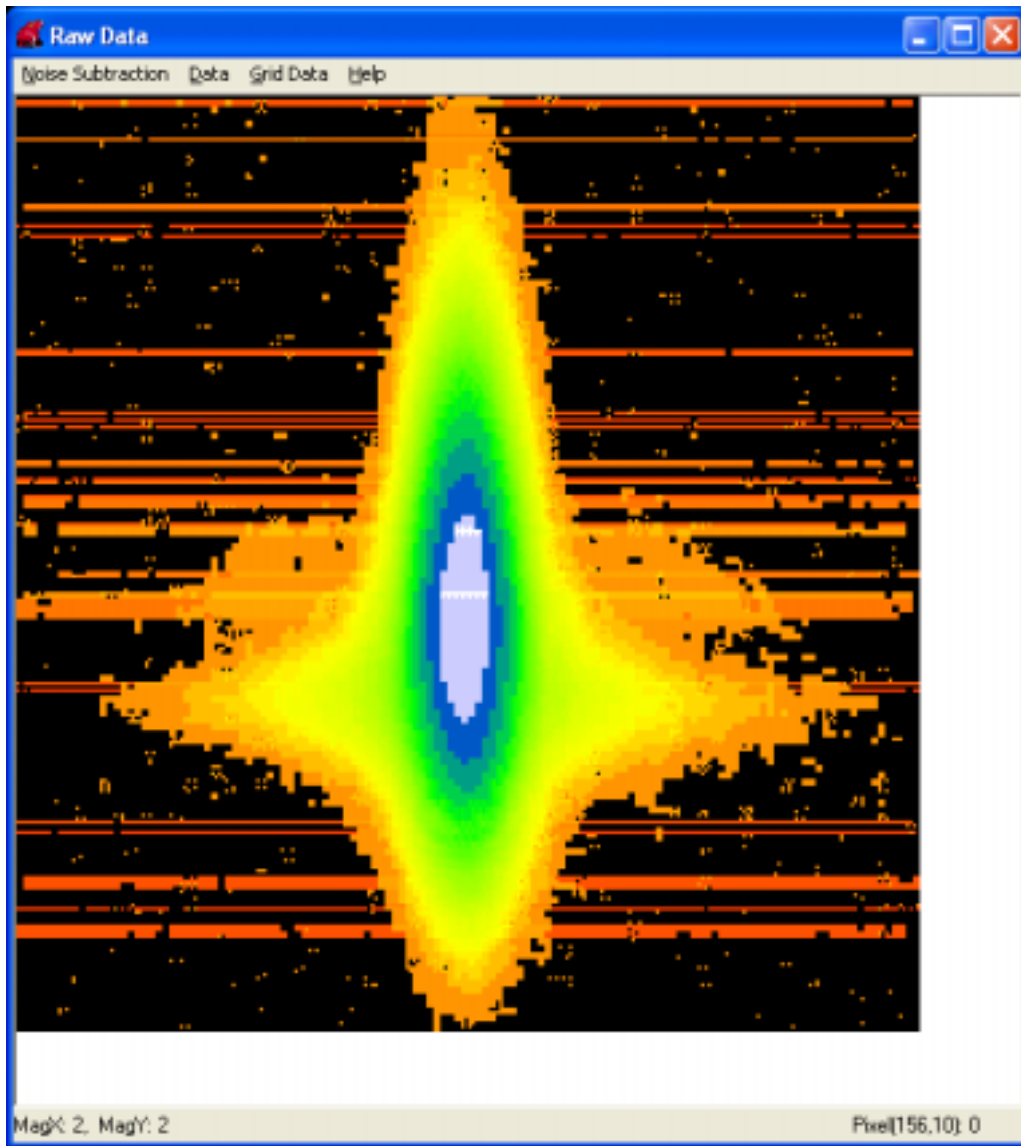
Try not to truncate any “real” FROG data. Now select **Data -> Extract**. Now our data set looks like this:



After data extraction.

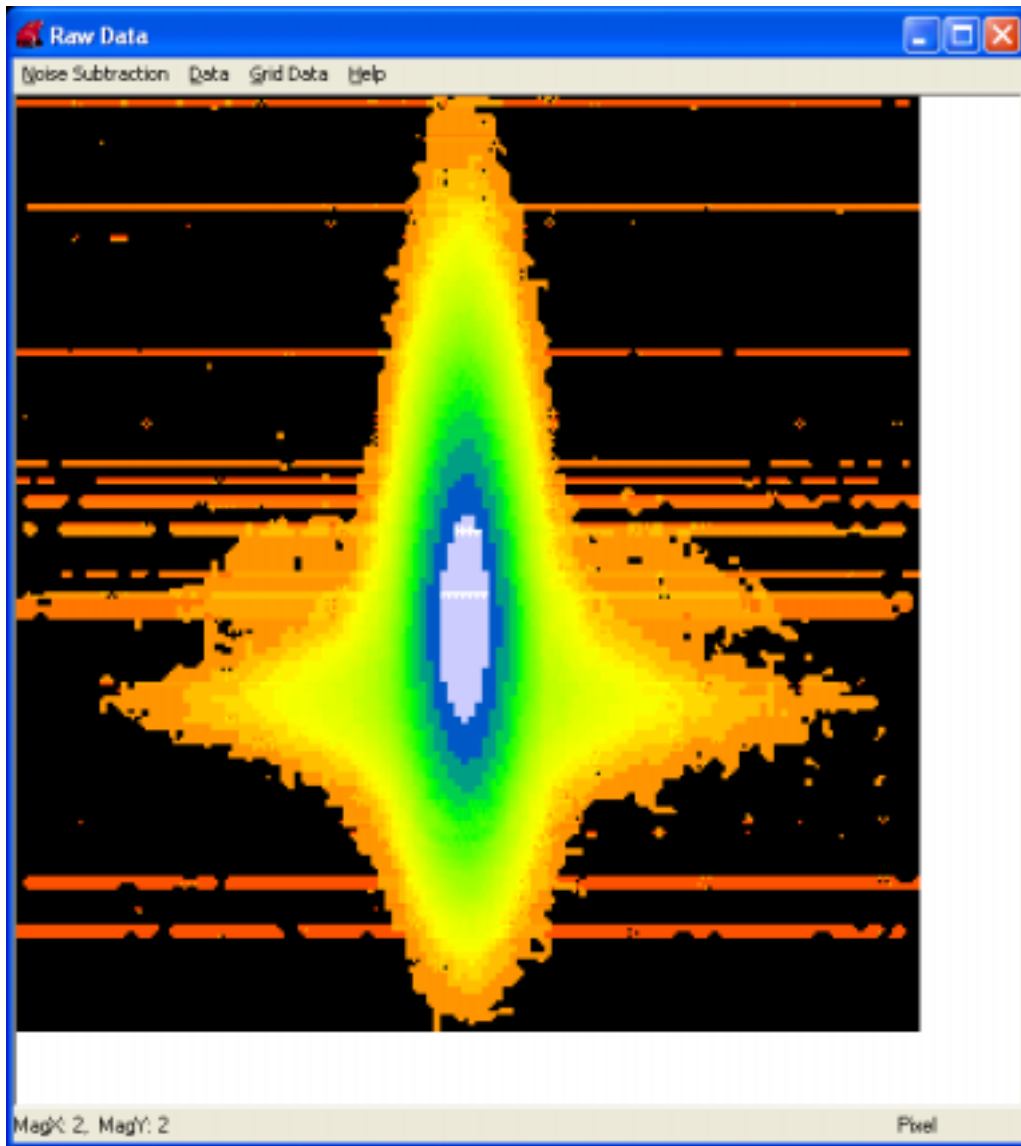
No we might have a better chance at full background subtraction. Let's try again.





After another full background subtraction.

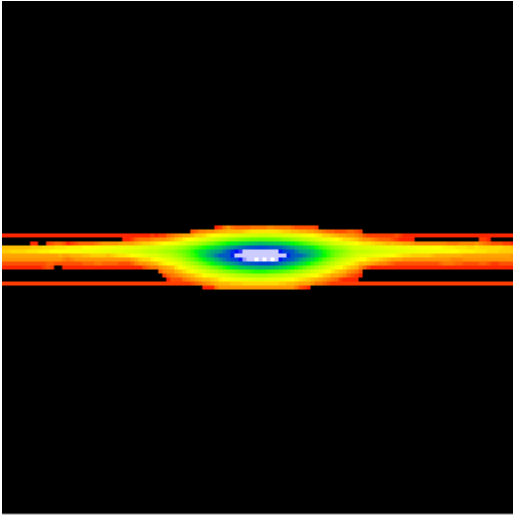
That looks much better, but we still have some noise floating around. This time, let's try **Noise Subtraction -> Cleanup Pixels**.



After a Cleanup Pixels.

That's a little better. We still have some noise stripes, but they are going to be very difficult to remove. We might try another Cleanup Pixels or two, but I'm afraid we're not going to do much better than this with this particular data set.

Let's try retrieving the data. Click **Grid Data**. With the grid size set to 128 points, here's what we see:



Gridded data.

The FROG trace is very wide and very short, in fact it appears to extend off the grid slightly. This is not good: in order to make sure we are satisfying the Nyquist sampling rate, we need to be sure that all the trace data is on the grid.

Now things get subtle.

## Sampling FROG Data

This brings up the issue of how to sample FROG data – how many points are enough, what grid size should I run at, etc. This gets subtle because the delay and frequency axes of the FROG trace are related through the Fourier Transform, and also because information about the pulse is in the time-frequency domain: if you want a finer time step, sometimes the right thing to do is use a larger frequency step!

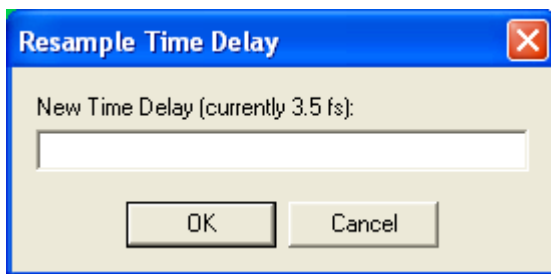
By default, the FROG software gridding algorithm will preserve the delay spacing of the data, and usually average over several points in the frequency direction. This is because in the old days of FROG, the typical way to work was to control delay with a stepper motor and resolve the FROG signal with a spectrometer that has a linear CCD array attached to it (not to imply that this sort of setup is out-of-date – although there have been clever variations on this setup proposed). So most data sets ended up with relatively fewer points in the delay direction (who wants to wait for the stepper motor to run?) and lots and lots of points in the frequency direction (high-resolution spectrometer).

Because of this default behavior (preserving the delay spacing), the FROG trace above is too wide. Apparently the delay spacing was a bit too small, stretching out

the trace in the horizontal direction. One option here is to use a larger grid size: go back and run the data with  $N = 256$ . This is fine; however, the algorithm slows down faster than  $N^2$ , so that this option will only take you so far. It's not advised to run at grid sizes like 1024 or 2048; it's just way too slow.

Note that the FROG trace above takes up very little of the area of the grid. If we could just squeeze it horizontally and expand it vertically a bit we'd be fine.

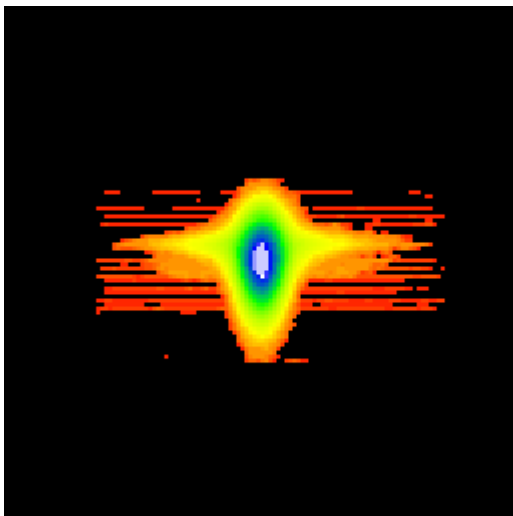
This is what the delay axis resampling is for. Start again from Ctrl+F, and get to the **Raw Data** window. Apply the noise subtraction and data extraction as before. Now select the **Data -> Resample Time Axis** menu item. The following dialog appears:



The resample time axis dialog box.

Here we can enter a desired delay axis spacing (in femtoseconds) and the software will re-interpolate the data onto this new grid. In our case, we want to compress the horizontal axis by a factor of 2 or 3, so I'll enter 10 fs as the new delay spacing.

You may want to try a couple more noise subtractions, and then grid the trace. It looks much better now:



Trace after resetting the delay axis.

Now the trace occupies only the center of the grid, the edges are (roughly) equidistant from the top/bottom and the left/right, etc. This looks like a very nicely fit trace. Of course, we still wish we didn't have all that noise, but it's very tricky to remove it.

Note that when we made the trace narrower, we also made it taller. The area of a gridded FROG trace is roughly proportional to the time-bandwidth product of the pulse, so as we change the aspect ratio the area of the trace is roughly preserved.

We made the delay step larger. That means the frequency step got smaller. It is possible that we have specified a frequency step that is smaller than that used to gather the experimental data. This will be noted in the header of the frog.dat output file (the same one that you can see from the **Log** menu item). Examining my log file, it says that there were "5.9 times too many wavelength points." This is good – the wavelength axis was sampled at almost 6 times too high a rate, a nice margin of safety. Before we changed the delay spacing, it said "17 times too many." If it ever says less than one times too many, then you'll need to go back and increase the frequency resolution of the experimental apparatus.

So how big of a grid size should you pick? We've been using 128 in this example. Retrieving a grid of size 128 is still pretty fast on most modern machines. It's also large enough that we don't have to spend a lot of time looking for the best delay spacing (usually). Nothing much is gained from going to a larger grid size, so long as all the data is contained on the grid. As the complexity (time-bandwidth product) of the pulse increases, a larger grid size is needed (because the trace area gets larger).

If the trace extends off the grid in the vertical direction, the experimental delay step size is too large. Because of the Fourier Transform relationship, the extent in frequency of the grid depends on the step size in the delay direction. If energy flows off the top or bottom of the grid, that means there is energy in the signal pulse at frequencies high enough to cause structure in the pulse at a finer scale than the delay spacing chosen.

When you can really understand that above paragraph, you're well on your way to understanding time-frequency analysis!

If the gridded trace stretches off the grid horizontally, increase the delay spacing. Watch the log file to make sure that the wavelength axis is not undersampled.

If the gridded trace stretches off the grid vertically, the data is undersampled in the time domain. Experimentally reduce the delay step.

#### Experimental Settings

When setting up an experimental apparatus for gathering FROG traces, there is no enforced relationship between the delay and wavelength axes. An apparatus could be set up with any arbitrary delay spacing and wavelength spacing. However, it is possible to undersample the trace data. For example, if the pulse to be measured has a FWHM of 10 fs, and the delay spacing of the apparatus is 150 fs, it's likely that you will not be able to retrieve the pulse. The symptom will be that no matter what values you use for grid size and delay spacing in the program, the full trace data will never fit on the grid.

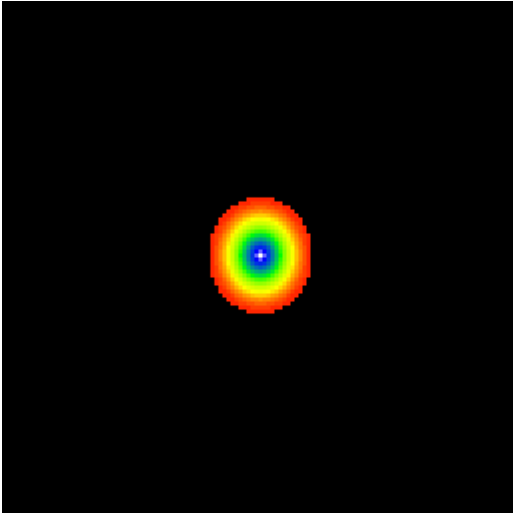
The issue of how densely to sample FROG data is quite subtle. Please check the literature for more detailed discussions (e.g., Chapter 10 in Trebino's book). But for "reasonable" pulses (fairly close to the time-bandwidth limit, not pathologically pulse-shaped pulses), if the data has 5-10 points within a FWHM in both the time and the frequency domain, it should be fine. For example, a 100 fs pulse with a 10 nm FWHM should be sampled at a delay spacing of 10-20 fs and with a wavelength resolution of about 1 - 2 nm. Pulses with higher time-bandwidth products will need finer resolution.

For a near-transform-limited pulse, it is not necessary to sample finer than 10 points per FWHM. E.g., a 100 fs pulse does not need to be sampled at 1 fs intervals!

## Calibration Errors

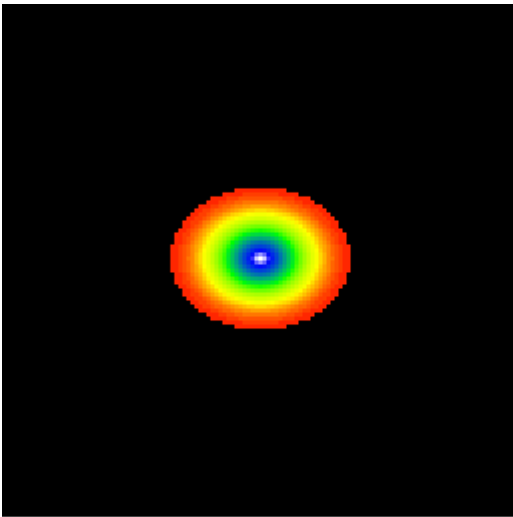
One of the most common problems with setting up a FROG device is getting the calibrations wrong. It's fairly difficult to measure these femtosecond pulses, they are after all on the edge of what is technologically achievable.

I often see traces that have fairly obvious calibration errors. For example, consider this SHG FROG trace on an  $N = 128$  grid:



SHG FROG trace,  $N = 128$  grid, with bad calibrations.

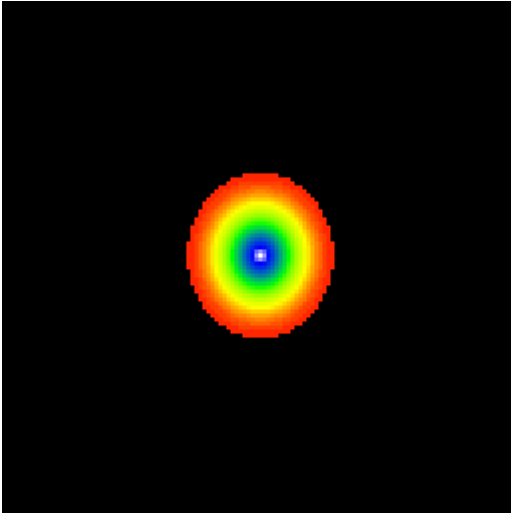
This trace is too small – the area is less than that of a transform-limited pulse. Here is a properly gridded transform-limited pulse:



Transform-limited pulse, SHG FROG,  $N = 128$ , correct calibrations.

You can see that its area is much larger than the first trace.

When the above incorrect trace is run through the algorithm, it retrieves to this:



Result of retrieving incorrect data.

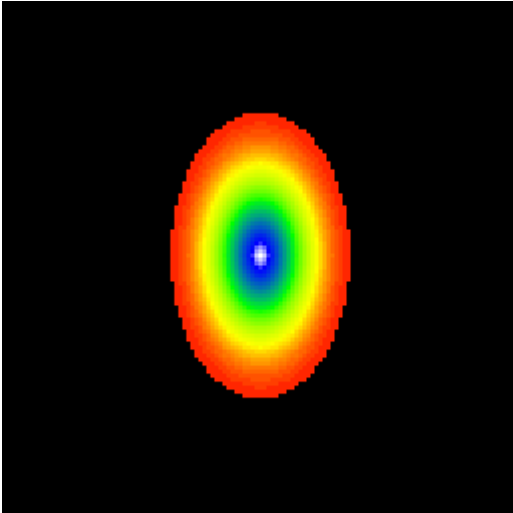
The algorithm does its best to match the target FROG trace. However, note that the size of the retrieved trace is quite different from the original (incorrect) trace. The FROG error was 0.014 (quite bad). The algorithm can only produce retrieved FROG traces that correspond to physically realizable electric fields – the FROG trace you see on screen is generated from an electric field. The original data in this case does not correspond to any physically realizable field, and therefore the algorithm cannot converge. No physical field can break the minimum time-bandwidth product specified by the Fourier Transform relationship (signal processing's analog to the Uncertainty Principle).

The problem in the above data was that the wavelength calibration was incorrect by a factor of 2. It's easy to get those terrible two's wrong.

The signature of too-small calibrations are a trace with area smaller than a transform-limited trace. The retrieved trace might be transform-limited or nearly transform-limited, but still much larger than the original trace.

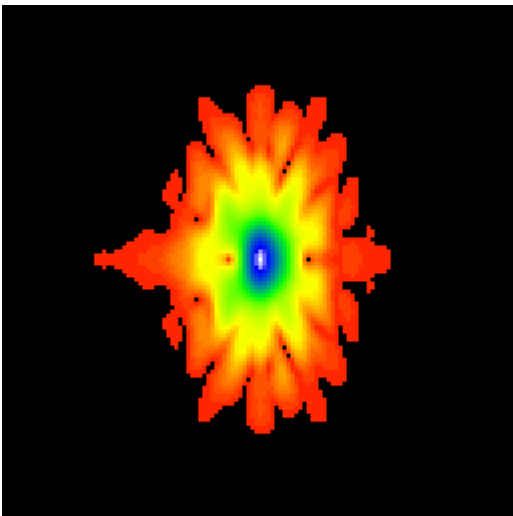
A similar thing can happen if the calibrations are too large. Here's a trace that I read in with the wavelength calibration double what it should be. It's a PG FROG trace at  $N = 128$ :





PG FROG,  $N = 128$ , wavelength calibration too large by factor of 2.

We can try to retrieve this pulse, but here is what we get:



Attempt to retrieve above pulse.

Notice how the retrieved pulse seems to “fold” back on itself, and develops a lot of “fingers” and other structure. This is the signature of a pulse that is too large in area. In PG FROG, the only way for a pulse to get a larger area than a transform-limited pulse is to acquire some phase distortions. But there are no phase distortions that will leave a PG FROG trace looking like a vertically-oriented oval. Linear chirp, for example, will make the trace tilt. So the algorithm is desperately trying to fill in the area of the original trace, but again no physical electric field can make that happen.

The signature of calibrations that are too large is the appearance of fingers, folds, etc. in the retrieved trace.

Note that in SHG FROG, matters are worse. A Gaussian transform-limited pulse has a FROG trace that is Gaussian in both directions, but *so does a Gaussian linearly-chirped pulse!* Therefore, for linearly-chirped pulses in SHG FROG, incorrect calibrations might do nothing more than change the amount of linear chirp that is measured. In general, retrieving an SHG FROG trace is a much tougher job than working with the other nonlinearities.

How then, to determine whether the calibrations are ok?

# The Marginals – Checking the Calibrations

## *FROG self-consistency tests*

### The Marginals

The marginals are one of the most powerful, but least-used, checks on FROG data. Mathematical details are left for the reader's own research (start with Chapter 10 in Trebino's book), but the basic idea is this. The mathematical form of the optical nonlinearities that are used to generate FROG traces imply a certain mathematical form for the trace data and its marginals (a marginal is the one-dimensional function that is obtained by integrating over one dimension of the FROG trace). It turns out that the marginals are related to other, independently-measurable quantities such as the spectrum and autocorrelation.

Therefore, if we measure a FROG trace (and therefore implicitly measure its marginals), and then we measure the appropriate quantities of the laser pulse (spectrum etc.) with an independent apparatus, we can check that the relationship between the two holds properly. If it does, we can be pretty sure that the calibrations of our apparatus are correct.

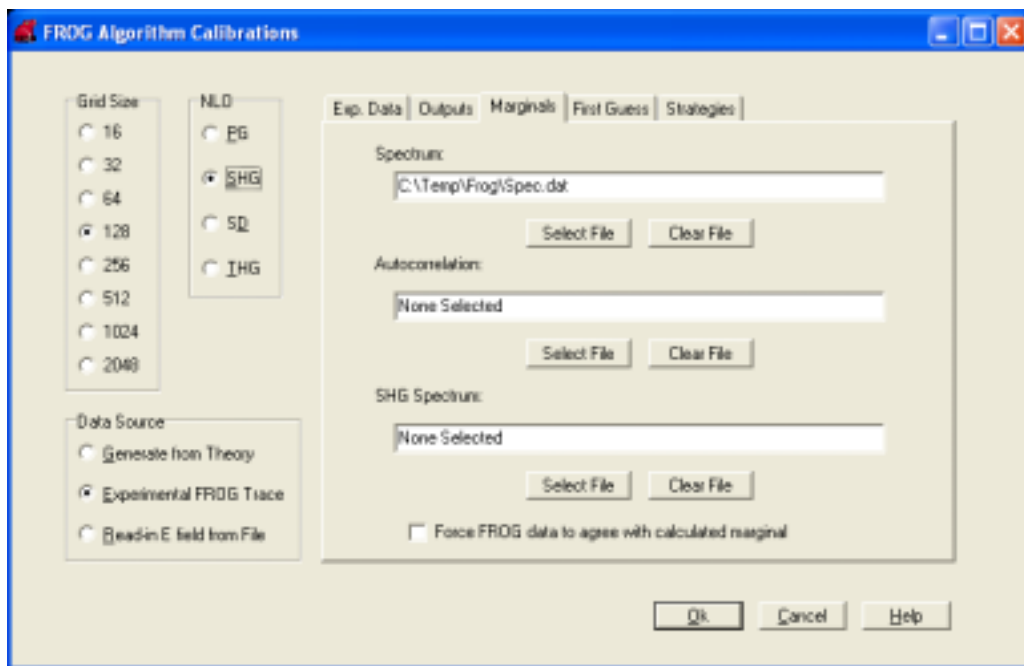
Let's use a specific example. The most convenient marginal is the SHG FROG frequency marginal. This is obtained by integrating an SHG FROG trace with respect to delay, leaving a one-dimensional function of frequency. It turns out that the frequency marginal must be identical to the autoconvolution of the intensity spectrum of the pulse being measured. Since the fundamental wavelength of the pulse is twice that of the SHG FROG signal, a different apparatus will most likely be used to measure it. If the frequency marginal and the autoconvolution of the fundamental agree with each other, either there is a pathological coincidence (less likely) or the apparatuses are calibrated properly (more likely).

For more information on the marginals, chase down the references in the literature (or better yet, do the math yourself!).

## Using Marginals in the Software

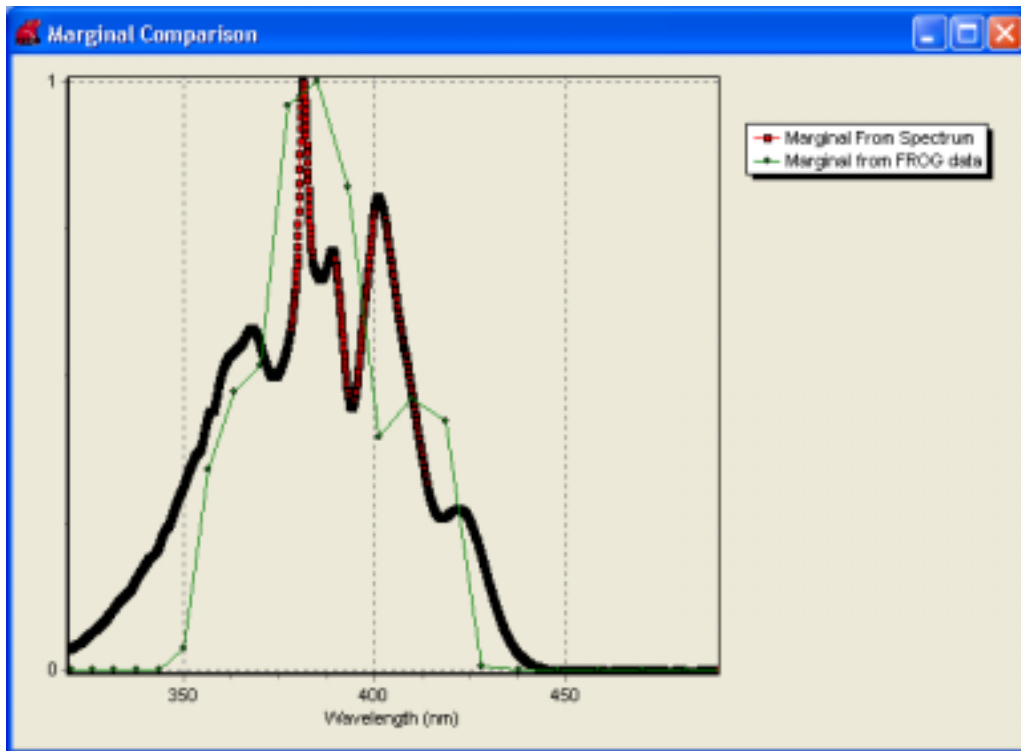
When working with experimental data, one of the tabs in the **FROG Algorithm Calibrations** window is called **Marginals**. This is where the extra data files are specified. Refer to the online help for the exact meaning and format of each data file.

For SHG FROG, we will need the spectrum of the fundamental pulse. Using the **Marginals** tab, select the file that contains the spectrum.



Selecting a spectrum with the marginal tab.

Now use the **Raw Data** window as before. When you grid the data, you will first see the marginal window. Here's an example of some poorly agreeing data:



Marginal Comparison Window

The red squares represent the autoconvolution of the fundamental spectrum of the laser pulse. The green dots represent the frequency marginal of the SHG FROG trace. These two functions should match (by the mathematics of second-harmonic generation), but here they bear only a passing resemblance. This indicates that one, or both, of the measurements are faulty.

At this point, one could right click the graph and select **Force Agreement**. This will adjust the FROG trace data so that the frequency marginal agrees with the autoconvolution of the spectrum. This is a dangerous game, however; it's much better to actually eliminate the errors in the experimental apparatus instead of trying to massage them away in software. Also, it assumes that the spectrum was measured correctly. Anyone who has worked with ultrashort pulses for a period of time knows that that's not an assumption that can always be made.

Note that the frequency marginal doesn't check the temporal coordinate; nevertheless it seems to be the most useful one in practice.

The other FROG nonlinearities use different sets of data to calculate their marginals. Check the online help for more information.

If you are having trouble getting convergence for experimental data,  
check the marginals!!

# Further Reading

## *Resources and Publications*

This manual barely scratches the surface of the FROG technique. You may want to consider these references for further reading:

### Main Reference

**Frequency-Resolved Optical Gating: The Measurement of Ultrafast Laser Pulses**, by Rick Trebino, (Kluwer Academic Publishers, 2000).

### Review Article

“Measuring ultrashort laser pulses in the time-frequency domain using frequency-resolved optical gating,” by Trebino et al., *Review of Scientific Instruments*, vol. 68, Sept. 1997, p. 3277.

### The Catalog Paper

“Comparison of ultrashort-pulse frequency-resolved-optical-gating traces for three common beam geometries,” by DeLong et al., *JOSA B*, vol. 11, Sept. 1994, p. 1595.

### SHG FROG

“Frequency-resolved optical gating with the use of second-harmonic generation,” by DeLong et al., *JOSA B*, v. 11, Nov. 1994, p. 2206.

### Marginals, Sampling Rates, etc.

“Practical Issues in Ultrashort-Laser-Pulse Measurement Using Frequency-Resolved Optical Gating,” DeLong et al., *IEEE-JQE*, vol. 32, July 1996, p. 1253.