DISSERTATION

EARLY TIME SOLVATION DYNAMICS OF ACETONITRILE AND METHANOL STUDIED BY BROADBAND FLUORESCENCE UPCONVERSION AND PHASE RETRIEVAL ANALYSIS

Submitted by
Benjamin Langdon
Department of Chemistry

In partial fulfillment of the requirements
For the Degree of Doctor of Philosophy
Colorado State University
Fort Collins, Colorado
Spring 2007

COLORADO STATE UNIVERSITY

December 22, 2006

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY BENJAMIN LANGDON ENTITLED EARLY TIME SOLVATION DYNAMICS OF ACETONITRILE AND METHANOL STUDIED BY BROADBAND FLUORESCENCE UPCONVERSION AND PHASE RETRIEVAL ANALYSIS BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

(Please print name under signature)

RANDY BARTELS

ELLIOT R. BERNSTEIN

Debbie C. Crans

C. MICHAEL ELLIOTT

Adviser

Nancy E. Levinger

Department Head

Anthony K. Rappé

ABSTRACT OF EARLY TIME SOLVATION DYNAMICS OF ACETONITRILE AND METHANOL STUDIED BY BROADBAND FLUORESCENCE UPCONVERSION AND PHASE RETRIEVAL ANALYSIS

Solvation dynamics of the small polar solvents acetonitrile and methanol were studied using broadband fluorescence upconversion techniques. Although broadband techniques were able to reproduce the solvent response functions of acetonitrile and methanol measured by traditional narrow bandwidth upconversion methods, they could not reproduce the *entire* time resolved fluorescence Stokes shift of Coumarin 153 in acetonitrile and methanol.

One advantage of using broadband upconversion techniques is seen in the increased sensitivity to small oscillations in the fluorescence frequency that are predicted and observed at relaxation times of less than one picosecond. These oscillations, attributed to librational motion, were observed in the time resolved fluorescence of Coumarin 153 in acetonitrile using broadband fluorescence upconversion. Some evidence for the presence of librational motion was seen from the early time dynamic fluorescence of Coumarin 153 in methanol, but it was not as conclusive as that seen in acetonitrile. Another advantage is that the solvation dynamics were readily reproduced using data from an upconversion experiment without tuning of the sum frequency crystal.

In addition to the inclusion of broadband detection, phase retrieval analysis techniques were adapted from optical physics to provide higher resolution dynamical information about the early time solvation dynamics of acetonitrile and methanol. The broadband nature of the Coumarin fluorescence posed an obstacle to directly calculating an instantaneous frequency from the retrieved temporal phase. To address this problem a

time-windowed intensity weighted average frequency was defined using the first time derivative of the temporal phase. In turn, a solvent response function was calculated using this average frequency. Applying this analysis to the results from a single-angle broadband fluorescence upconversion experiment showed favorable results for producing a solvent response function for acetonitrile. The results from applying phase retrieval analysis to the methanol experimental data were less successful. The problem with application to the methanol data was attributed, in part, to the higher bandwidth of its fluorescence. The increased bandwidth and resulting decreased coherence time caused problems with the calculation of the time-windowed intensity weighted average frequency.

Benjamin Langdon
Chemistry Department
Colorado State University
Fort Collins, Colorado 80523
Spring 2007

Table of Contents

Chapter 1

Introduction

Motivation

The study of polar solvation dynamics is essential for understanding the kinetics of chemical reactions in liquid solution and more specifically that of electron transfer reactions. Much biologically important chemistry proceeds by electron transfer reactions whose rates are, in many cases, dominated by solvent reorganization.

Time resolved fluorescence Stokes shift experiments can be used to study solvent motion influencing charge transfer processes. This experiment uses a fluorescent solute to both perturb and probe the electrostatic environment created by the local solvent molecules. Optical excitation of the solute instantaneously increases its dipole. Then, as the dye fluoresces, its peak fluorescence wavelength changes according to the relaxation of the solvent environment.

Sub-picosecond resolution of the dynamic fluorescence Stokes shift requires nonlinear optical gating methods such as fluorescence upconversion.[1] Conventionally, the upconverted fluorescence signal is collected one wavelength at a time by a monochromator with a photomultiplier tube over a series of delay times after the excitation of the probe. Each detected wavelength requires tuning of the nonlinear optical crystal. Then, these individual fluorescence decays are used to reconstruct the time resolved fluorescence.[2-5] This methodology utilizing photomultiplier tubes for detection arose from the necessity to use very sensitive photodetectors due to the weak

nonlinear optical signal generated by the upconversion process. As array detection (via charge coupled devices and photodiode arrays) has become more sophisticated and more sensitive, it is now possible to record an entire time resolved spectrum at once.

At the same time, laser technology has improved so that sub 30 fs pulses are routinely available from entirely solid-state oscillators. However, these shorter pulses come with the increased complexity accompanying the required increased bandwidth. For laser pulses approximately 100 fs or longer, linear optical dispersion caused by transmissive optics can largely be ignored. The bandwidth of the pulses required to resolve the fastest solvation dynamics requires that special care be taken to account and possibly compensate for linear dispersion and other optical phenomena that are more prevalent in broadband pulses. To this end, we can take advantage of advances in ultrashort pulse diagnostics such as frequency resolved optical gating (FROG) and optical phase retrieval.

Polar solvation dynamics

The study of polar solvation dynamics requires rapid control of the electrostatic environment of the polar solvent under investigation as well as a method for tracking the solvent's reaction to changes in this environment. One method that can achieve this control is application of a time varying external electric field applied across a thin layer of polar solvent via parallel plate capacitors. As the electric field is varied, the capacitance of the dielectric solvent between the plates is measured. The frequency of the time dependent field can be systematically varied to determine at which frequency the polar solvent can no longer effectively act as a dielectric medium to reduce the electric field strength between the plates of capacitor. This technique is useful up to frequencies

of hundreds of gigahertz, which is the limit of the oscillating frequency that can be produced by modern electronics. This ultimately results in resolution of molecular motion on the tens to hundreds of picoseconds time scale.

Solvent dynamics faster than what can be measured using the method described above can be observed using time resolved fluorescence Stokes shift experiments.[1] This experimental technique uses an optically excited probe molecule to both perturb the electrostatic environment of the solvent as well as track the reaction of the polar solvent. Time resolved fluorescence Stokes shift (TRFSS) relies on a fluorescent probe molecule that exhibits a large change in dipole moment upon excitation.[6-9] This large increase in dipole moment changes the electrostatic environment experienced by the polar solvent. After excitation, the fluorescence frequency of the probe molecule depends on the orientation of the polar solvent molecules around it providing a convenient physical property with which to monitor the progress of the reorientation of the solvent.[2-4]

Before optical pumping, the probe molecule (large gray ellipse in Figure 1.1) is solvated by the equilibrated polar solvent. A representation of the potential energy surface of the system in the ground state is depicted by the S_0 curve. The S_0 and S_1 curves are a 2-D crosscut through the N-dimensional potential surface formed by all of the nuclear degrees of freedom of the solvent. The horizontal displacement of the potential minimum of each surface illustrates that each minimum exists at a different point along the solvent coordinate.

The vertical arrow labeled Pump Pulse in Figure 1.1 represents excitation of the probe molecule. The pump pulse causes a Frank-Condon transition in the probe/solvent system that leaves the nuclear positions of the solvent unchanged because the excitation

event happens too rapidly for the nuclear degrees of freedom to react.[2, 3] As the nuclear degrees of freedom of the solvent react to the new dipole through rotation and translation, the free energy of the entire system decreases and the probe fluorescence frequency decreases (right panel).

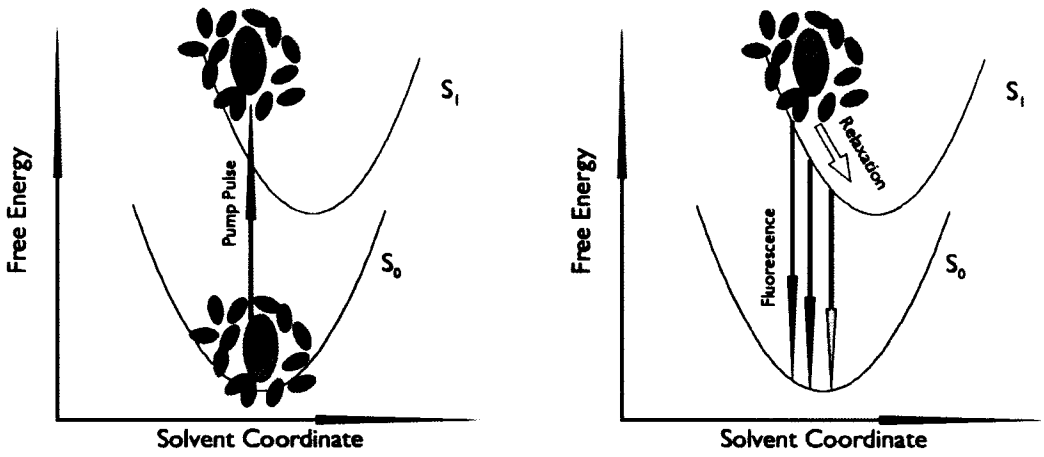


Figure 1.1 The left panel shows the vertical transition from the S_0 to the S_1 potential energy surfaces of the probe/solvent system. The right panel shows the relaxation of the system resulting in the decrease of the probe fluorescence frequency.

The primary function calculated from TRFSS experiments is the solvent response function, *C(t)*. The *C(t)* tracks the progress of the solvent relaxation process and is calculated from the time resolved dynamic fluorescence

$$C(t) = \frac{\nu(t) - \nu(\infty)}{\nu(0) - \nu(\infty)}$$

Equation 1.1

where $\nu(t)$ is some spectral measure, e.g. peak frequency or average frequency.

Time resolved fluorescence Stokes shift experiments require molecular probes with specific properties. The probe must have a small dipole in the ground state and a

large dipole in the excited state. The small dipole in the ground state allows the solvent to achieve an unperturbed equilibrium about the solute. After excitation, the solute molecule must have a large enough increase in dipole to perturb the surrounding polar solvent. At room temperature, most polar solvents will relax on time scales faster than a few hundred picoseconds. Thus, the excited state lifetime of the probe molecule should be at least an order of magnitude longer than this solvent relaxation time. Additionally, the probe must not be significantly vibrationally coupled to the solvent. A probe with significant vibrational coupling to the solvent could transfer excess vibrational energy to the local solvent environment thereby obscuring the dynamics attributable solely to dipolar relaxation.

Spectroscopic Features of Coumarin 153

Coumarin 153 (C153), shown in Figure 1.2, has been shown to be an excellent solvation dynamics probe molecule.[10-14] C153 undergoes a large increase in the magnitude of its dipole moment after the $S_0 \leftrightarrow S_1$ transition. The ground state dipole has been measured to be 6.55 ± 0.01 D while the excited state dipole has been measured to be 14-16 D depending on solvent.[10] The $S_0 \leftrightarrow S_1$ transition has also been shown to be free of interference from any competing electronic transitions[10] so excitation near the absorption maximum should result in only one excited state species.[15]

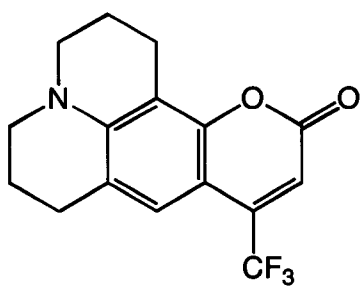


Figure 1.2 Chemical structure of Coumarin 153

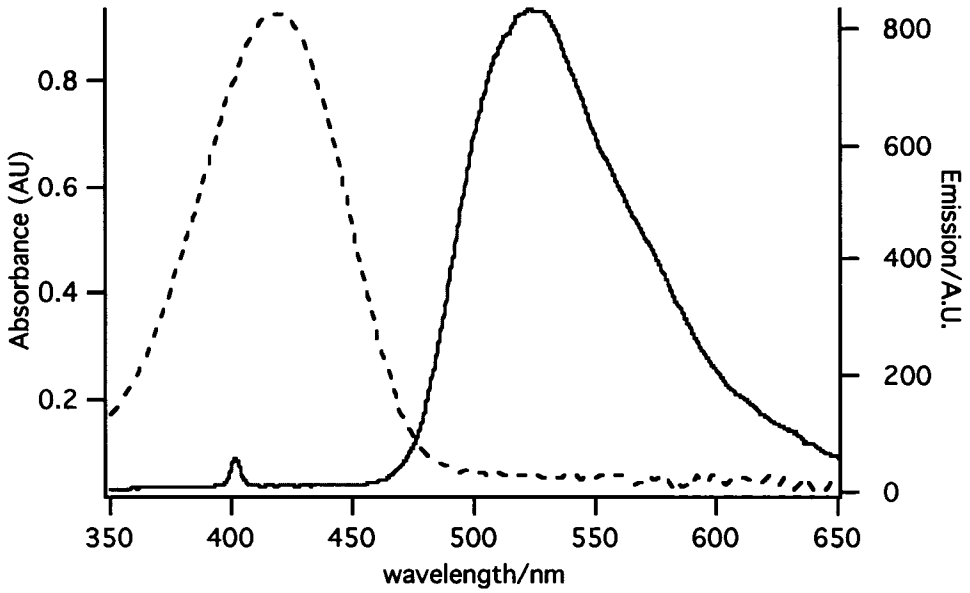


Figure 1.3 Absorption (dashed line) and emission (solid line) spectra for Coumarin 153 in acetonitrile.

Dipolar Solvent Response to an Instantaneous Electrostatic Perturbation

The response of small dipolar solvents such as acetonitrile and methanol can be roughly divided in to two time regimes. The first is the so-called inertial response, followed by the diffusive response.[16] The inertial response is characterized by the under-damped motion of solvent molecules within the first one to two shells around the solute. For small solvents such as acetonitrile and methanol, molecular dynamics simulations show that the *C(t)* has an approximately Gaussian shape at times within the inertial response regime, up to 1 ps.[17-19] Furthermore, inertial relaxation is the most important mechanism in the response of small solvent molecules such as acetonitrile and methanol and can account for as much as 90% of the overall relaxation.[10, 11, 17-20] The inertial motion in small dipolar solvents has been demonstrated to be largely

rotational, as opposed to translational, in nature by molecular dynamics simulations by Ladanyi and Maroncelli. [18]

Diffusive motions are responsible for the remainder of the relaxation in small polar solvents and are characterized by their longer time scales. The diffusive tail of the solvent response function can be single or multi-exponential depending on the number of relaxation modes present in the solvent. As the name implies, diffusive relaxation propagates outward from the solute and is comprised of both rotations and translations of the solvent.[18, 21, 22]

Wave Description of an Optical Pulse

In order to quantitatively describe the nonlinear optical processes used in this work, a mathematical description of the electric field of an optical pulse is necessary. The most convenient description to use for this application is the complex representation of the electric field, Equation 1.2

$$E(t) = \frac{1}{2}\mathcal{E}(t)e^{i\phi(t)}e^{i\omega_0 t} + \text{c.c.}$$

Equation 1.2

where $\mathcal{E}(t)$ is the amplitude, $\phi(t)$ is a function describing the phase and ω_0 is the carrier frequency of the wave. Fourier transforming $E(t)$ yields the complex spectrum:

$$\mathcal{F}\{E(t)\} = \tilde{E}(\Omega)$$

Equation 1.3

From here some simplifications can be made. A complex envelope function, $\tilde{\mathcal{E}}(t)$, can be defined as

$$\tilde{\mathcal{E}}(t) = \mathcal{E}(t)e^{i\phi(t)}$$

Equation 1.4

Neglecting the complex conjugate and substituting into Equation 1.2 gives

$$E(t) = \frac{1}{2}\tilde{\mathcal{E}}(t)e^{i\omega_0 t}$$

Equation 1.5

This description using the complex envelope is valid as long as the bandwidth, $\Delta\Omega$, of the signal is a small fraction of the carrier frequency, i.e., $\frac{\Delta\Omega}{\omega_0} \ll 1$. In the case of the fluorescence signals studied in this work, the maximum bandwidths measured are on the order of 3 x 10^{13} Hz with carrier frequencies of approximately 6 x 10^{14} Hz, giving a ratio of 0.05. Further simplification comes from treating the carrier frequency as a constant offset and only being concerned with changes to the complex envelope. A Fourier transform of the complex envelope will result in a spectrum distributed about zero instead of the carrier frequency ω_0.

Fluorescence Upconversion

One tool for measuring a dynamic fluorescence signal on pico- and subpicosecond time scale is fluorescence upconversion.[2, 7, 23] In general, a fluorescence signal is gated using a shorter light pulse (gate pulse) and a nonlinear optical crystal. The temporal and spatial overlap of the dynamic fluorescence signal with the gate pulse within the nonlinear optical crystal causes the generation of a sum frequency signal. The sum frequency signal is the sum of the instantaneous frequency of the dynamic fluorescence signal and the gate pulse. The gate pulse will only upconvert the portion of

the fluorescence signal that is present in the nonlinear optical crystal at the same time. Varying the delay between the start of the fluorescence signal and the gate pulse allows the gate pulse to systematically upconvert various portions of the fluorescence signal thereby mapping its dynamical behavior.

This nonlinear optical process has been utilized to reconstruct the time resolved fluorescence from fluorescent dyes in solution through a procedure called spectral reconstruction.[10] In this procedure, 10-15 fluorescent decays are collected at various emission frequencies by tuning the monochromator to the desired upconversion emission frequency. The upconversion frequency is the sum of the desired emission frequency plus the frequency of the gate pulse. The delay between the excitation and gate pulses is set to ensure sufficient signal is detected at the desired upconverted emission frequency. With the delay stage stationary, the angle of the nonlinear optical crystal is adjusted to maximize the upconverted signal. Finally, the delay between the excitation and gate pulses is set so that the gate pulse arrives at the nonlinear optical crystal before the emission from the fluorescent sample. From here the delay can be varied systematically to record the upconversion intensity at the chosen frequency.

After several of these decays are collected at a range of frequencies, the decays are fit to a sum of exponentials (usually also convoluted with a Gaussian that represents the laser pulse). The fitted decays are then normalized relative to their corresponding frequency intensities in the steady state fluorescence spectrum from excitation at the same wavelength as the excitation pulse. [2-4]

Sum frequency generation (SFG) is the nonlinear optical phenomenon by which two optical waves combine in an appropriate nonlinear optical medium to produce a third

wave. Nonlinear processes, such as second harmonic and sum frequency generation, are the result of a power dependent nonlinear polarization response of the optical medium.As previously mentioned, generation of a sum frequency signal requires temporal and spatial overlap of the fluorescence signal and a gate pulse within a nonlinear optical crystal. The sum frequency of the two input pulses is generated by the nonlinear polarization of the optical medium by the two optical pulses. In this process the polarization of the optical medium produces oscillations at the sum frequency of the input light.

The efficiency of the sum frequency process depends on the frequency and propagation vectors of the input beams. When the frequencies of the input beams and their propagation vectors within the nonlinear optical crystal are optimized, the beams are said to be phase matched and the sum frequency or second harmonic signal is generated most efficiently. This condition can only be met over a small frequency range, so optimal upconversion efficiency ($\eta(\Delta k)$) is only available for a portion of the fluorescence signal. The efficiency dependence on the propagation vectors is

$$\eta(\Delta k) \propto \mathrm{Sinc}^2(L\Delta k)$$

Equation 1.6

where

$$\Delta k = |\mathbf{k}_{\mathrm{SFG}} - \mathbf{k}_{\mathrm{g}} - \mathbf{k}_{\mathrm{fl}}|$$

Equation 1.7

and

$$|\mathbf{k}| = \frac{\omega n(\omega)}{c_0}.$$

Equation 1.8

$n(\omega)$ is the frequency dependent refractive index and c_0 is the speed of light.[24, 25]

Frequency Resolved Optical Gating

An extension of the fluorescence upconversion technique mentioned above is Frequency Resolved Optical Gating (FROG). Originally this technique was developed to measure the duration of laser pulses that last for only a few femtoseconds and to determine temporal electric field of the pulse.[26-28] In general FROG methods utilize some form of nonlinear optical gating (e.g. second harmonic generation or polarization gating) to temporally resolve ultrashort optical pulses.[29]

Instead of detecting the frequency-integrated intensity as in simple pulse autocorrelation, the FROG technique frequency resolves the nonlinear optical signal to provide simultaneous measurement of pulse intensities as a function of both time and frequency. Measurement of time and frequency resolved nonlinear optical signals along with knowledge of the nonlinear process that produces the signals allows for the retrieval of the electric field of the input pulses through the phase retrieval algorithm.[26-28]

The simplest of the FROG techniques utilizes second harmonic generation (SHG) to gate a pulse with a copy of itself.[26-28] In SHG FROG, an optical pulse is split with a beam splitter, and then the two resulting beams are spatially overlapped in an SHG crystal to produce a pulse with double the frequency. The relative delay of the two beams is varied and the second harmonic signal is frequency resolved with a prism or dispersive grating to produce a spectrogram of the resulting pulse. The spectrogram is a two-dimensional matrix of signal intensities with relative time delay values along the horizontal dimension and signal frequency values along the vertical dimension. Figure 1.4 shows a spectrogram of a laser pulse from a titanium-sapphire (Ti:sapph) laser oscillator with central wavelength of 808 nm, full-width at half-maximum (FWHM), temporal width of 37 fs, and spectral width of 22 nm (FWHM).

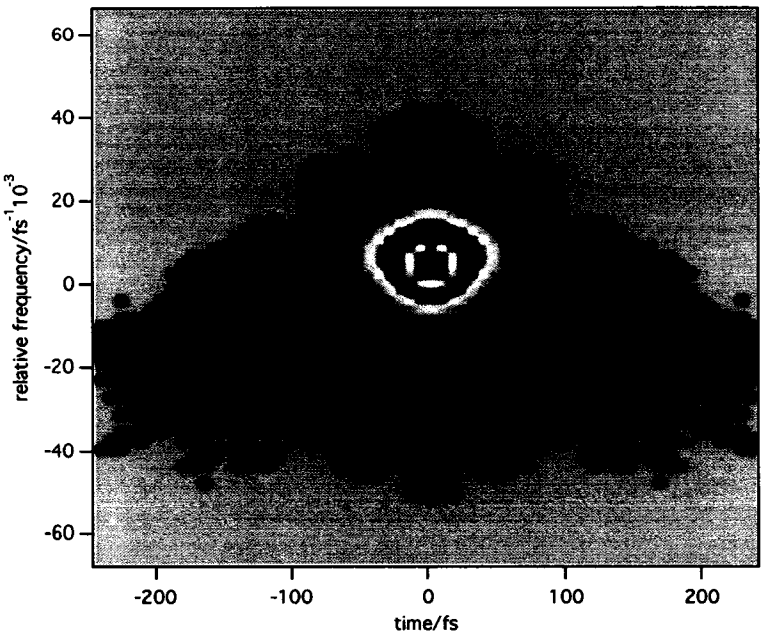


Figure 1.4 Spectrogram of a 37 fs (FWHM) laser pulse from a Ti:sapph oscillator. Higher intensities are blue while lower intensities are black.

Phase Retrieval Algorithm

While recording a spectrogram of a pulse is somewhat useful, retrieval of the temporal magnitude and phase of the pulse allows for its complete characterization. Kane, Tebino and coworkers[29] demonstrated that knowledge of the nonlinear gating process employed for the temporal gating can be used to reconstruct, or retrieve, the input pulse through an iterative deconvolution process called phase retrieval. What follows is a summary of methods developed by Kane, Tebino and coworkers for phase retrieval of laser pulses from SHG FROG spectrograms.[29]

The phase retrieval algorithm follows a relatively simple scheme for reconstructing the electric field of an input pulse from the measured spectrogram. The first step is to calculate the electric field of the gated output signal from a trial input pulse and the mathematical representation of the nonlinear optical gating process. The second step is to apply the real measured spectrogram to the one calculated from the trial pulse.

Last, decomposing the spectrogram into its component vectors generates a new trial pulse. The newly created trial pulse is then used as the start of the next iteration of the algorithm.

The simplest of the nonlinear gating processes is second harmonic generation whose expression is

$$\tilde{\mathcal{E}}_s(t,\tau) \propto \tilde{\mathcal{E}}_t(t) \times \tilde{\mathcal{E}}_t(t-\tau)$$

Equation 1.9

where $\tilde{\mathcal{E}}_s(t,\tau)$ is the electric field of the SHG signal parameterized by the relative delay between the two copies of the pulses arriving at the SHG crystal and $\tilde{\mathcal{E}}_t(t)$ is the electric field of the trial pulse. The next step is to perform a 1-D Fourier transform of $\tilde{\mathcal{E}}_s(t,\tau)$ at constant values of τ to yield the function $S_s(\omega,\tau)$.

$$\begin{aligned} S_s(\omega,\tau) &\propto \mathcal{F}\{\tilde{\mathcal{E}}_s(t,\tau)\}_\tau \\ &\propto \int_{\infty}^{\infty} \tilde{\mathcal{E}}_s(t,\tau)e^{-i\omega t}dt \end{aligned}$$

Equation 1.10

The next step is to apply the measured spectrogram, $I(\omega,\tau)$, replacing the magnitude of $S_s(\omega,\tau)$ with the square root of the measured spectrogram.

$$S_s(\omega,\tau)^\dagger = \sqrt{I(\omega,\tau)}\frac{S_s(\omega,\tau)}{|S_s(\omega,\tau)|}$$

Equation 1.11

Next, a new $\tilde{\mathcal{E}}_s(t,\tau)$ is produced from the inverse 1-D Fourier transform of $S_s(\omega,\tau)^\dagger$

$$\tilde{\mathcal{E}}_s(t,\tau)^\dagger \propto \mathcal{F}^{-1}\{S_s(\omega,\tau)^\dagger\}_\tau$$

Equation 1.12

The last step is to produce a new trial electric field from $\tilde{\mathcal{E}}_s(t,\tau)^{\dagger}$.

There are a variety of methods to accomplish this, but only the one used in this work will be discussed here.[29] In practice, $\tilde{\mathcal{E}}_t(t)$ and $\tilde{\mathcal{E}}_s(t,\tau)$ are sampled at regular intervals of t and τ producing in a vector for $\tilde{\mathcal{E}}_t(t)$ and a matrix for $\tilde{\mathcal{E}}_s(t,\tau)$. Recall that $\tilde{\mathcal{E}}_s(t,\tau)$ is produced by

$$\tilde{\mathcal{E}}_s(t,\tau) \propto \tilde{\mathcal{E}}_t(t) \times \tilde{\mathcal{E}}_t(t-\tau)$$

Equation 1.13

This allows matrix decomposition methods to be used to find the trial pulse for the next iteration of the phase retrieval algorithm. By defining a matrix $\hat{O}$ by discretely sampling $\tilde{\mathcal{E}}_s(t,\tau)$ over the relevant ranges of t and τ, singular value decomposition can be used to find the orthogonal square matrices $\hat{U}$ and $\hat{V}$, and the diagonal matrix $\hat{W}$ that best satisfy the generalized algebraic expression

$$\hat{O} = \hat{U} \times \hat{W} \times \hat{V}^T$$

Equation 1.14

The columns of $\hat{U}$ and $\hat{V}$ represent pairs of vectors ($\tilde{\mathcal{E}}_t(t)$ and $\tilde{\mathcal{E}}(t-\tau)$), which are possible solutions to Equation 1.14. The matrix $\hat{W}$ is a diagonal matrix whose elements are the relative weight factors of the pairs of column vectors from $\hat{U}$ and $\hat{V}$. The algorithm selects the column vector from $\hat{U}$ with the largest weighting factor from $\hat{W}$, called the principal singular vector, as the trial electric field for the next iteration. As the algorithm iterates, the trial electric field is improved and the weighting factor of the principal singular vector pair becomes much larger than that of any other vectors. The ultimate solution for the algebraic expression (Equation 1.14) that represents a perfect

noiseless FROG measurement would be a matrix $\hat{O}$ of rank one, meaning that the matrix $\hat{W}$ would contain only one nonzero element or weighing factor and thus there would be only one possible pair of column vectors from $\hat{U}$ and $\hat{V}$ that would satisfy Equation 1.14. Furthermore, SHG-FROG places the requirement that $\hat{U}$ and $\hat{V}$ be identical because they are, in fact, representations of the same pulse.

Cross Correlation Frequency Resolved Optical Gating and Modifications to the Phase Retrieval Algorithm

The SHG-FROG technique has proven to be a valuable asset for complete characterization of ultrashort laser pulses,[26-28] but some modification to the methodology is required for application to study of chemical systems where we wish to measure the time resolved fluorescence Stokes shift. Modifying the fluorescence upconversion technique to use array detection allows for the collection of a spectrogram of a dynamic fluorescence signal. In contrast to the SHG-FROG that splits a single pulse so that it can gate itself and produce an SHG signal, cross correlation FROG (XFROG) uses a reference pulse to gate the dynamic fluorescence signal to produce a sum frequency (SFG) signal[26-28] For this work, the reference pulse was the compressed pulse from a Ti:sapph laser that had been previously characterized using a commercially produced FROG.

The modification to the phase retrieval algorithm reflects the change of the input pulses. The delay parameterized SFG signal is now produced by the following equation

$$\tilde{\mathcal{E}}_s(t,\tau) \propto \tilde{\mathcal{E}}_g(t) \times \tilde{\mathcal{E}}_f(t-\tau)$$

Equation 1.15

where $\tilde{\mathcal{E}}_g(t)$ is the electric field of the independently characterized reference pulse or gate pulse that is never modified in the algorithm. $\tilde{\mathcal{E}}_f(t-\tau)$ is the trial electric field of the dynamic fluorescence we are attempting to characterize. When performing the singular value decomposition, because the input fields are distinct, $\hat{U}$ and $\hat{V}$ cannot be identical. $\hat{U}$ represents the fluorescence electric field $\tilde{\mathcal{E}}_f(t-\tau)$ and $\hat{V}$ represents the gate pulse electric field $\tilde{\mathcal{E}}_g(t)$. The principal singular vector of $\hat{U}$ is then used as the trial fluorescence electric field for the next iteration of the algorithm.

Chapter 2
Experimental Materials and Methods

Sample preparation

All solution samples were made using the following procedure. Approximately one to three mg of the Coumarin dye (C153 Exciton) were added to a 25 ml glass vial, the neat solvent (Fischer, HPLC Grade) was then added to the vial resulting in a concentration of 0.1-0.4 mM. The solution was then sonicated for 10 minutes to ensure that the dye was completely dissolved. All absorption spectra were collected with an HP 8452A diode array absorption spectrometer. Steady state fluorescence spectra were collected using an ISA-Spex Fluorolog fluorescence spectrometer with excitation wavelength to match the excitation source of the time resolved fluorescence instrument.

Additional steady state fluorescence measurements were made of the dye emission directly from the optical flow cell in the time resolved experiment using an Ocean Optics mini-spectrometer and the same excitation source used in the time resolved experiment. After filling the flow cell apparatus and starting the paristaltic pump, the approximately 400 nm excitation pulse was focused on the flow cell. Then the optical fiber input from the Ocean Optics spectrometer was held close to the effluent spot in the cell while the spectrum was acquired by the Ocean Optics software.

Description of the time-resolved fluorescence instrument

Figure 2.1 depicts the instrument used for this work. Pulses were created with a Ti:sapphire oscillator (KMLabs) operating with a peak wavelength of 804-810 nm with approximately 40 fs pulse width FWHM measured by SHG FROG (Swamp Optics). The pulses from the oscillator were frequency doubled to 402-405 nm and pulse width of 60-90 fs FWHM (measured by home built cross-correlation SFG FROG) with a 0.25 mm thick type II beta-barium borate (BBO) nonlinear optical crystal (Casix). The horizontally polarized ~800 nm fundamental pulses were focused onto the BBO (BBO1 in Figure 2.1) with a 10 cm lens (L1). The frequency doubled pulses were reflected with a dichroic mirror (DM) to separate them from the residual fundamental beam. The frequency doubled beam that produces the excitation or pump beam for the time-resolved fluorescence experiments was collimated with a 10 cm focal length lens (L2). Because of the type II nonlinear interaction with the BBO crystal, the pump beam emerges vertically polarized. The polarization was rotated with a half waveplate (WP1, Meadowlark) to maximize throughput of the beam in the subsequent prism pair compressor. To compensate for group velocity dispersion due to transmissive optics in the beam path, the horizontally polarized pump beam was directed through a prism pair compressor (quartz, KM Labs) (PPC1) whose prism separation was 63 cm. This separation was determined empirically (see Apendix D) by minimizing the cross-correlation FROG signal duration at the SFG crystal. Immediately before impinging on the sample, the pump pulse traverses second half waveplate (WP2) for control of the final polarization and was focused onto the sample flow cell with a 10 cm focal length lens.

The fluorescence from the sample was collected with an ellipsoidal reflector (ER). The sample sits in one focus of the ellipsoidal reflector allowing it to relay the sample fluorescence to the BBO (BBO2) sum frequency generation crystal placed at the other focus of the ellipsoidal reflector.

The residual fundamental pulses at 800 nm forming the gate beam were collimated with a 10 cm focal length lens (L3). Next, the gate beam was compressed with a prism pair compressor (PPC2) with separation of 154 cm to compensate for all of the accumulated group velocity dispersion before the SFG crystal (BBO2). This separation distance was determined by minimizing the intensity auto-correlation signal width right before the SFG crystal. After compression, the gate beam path length was variably changed with the use of a movable mirror stage translated by a stepper motor and controlled via the data acquisition computer using a custom LabView program. The stepper motor translated stage provided time resolution of 16.68 fs. After exiting the variable delay stage, the polarization of the gate beam was rotated to be vertical by passing it through a half waveplate (WP3). Finally, the beam was focused onto the SFG crystal (BBO2).

The fluorescence and gate beams intersected at the SFG crystal with an angle of approximately 34°. The emerging fluorescence, gate and sum frequency signal were collimated with a 20 cm focal length quartz lens (L6). The residual fluorescence and gate beams were directed onto the entrance port of the monochromator and used for alignment of the ultra-violet SFG signal. The visible beams were then blocked to prevent interference with the detection of the SFG signal.

Initially, the instrument was constructed with multi-stack dielectric mirrors

purchased from New Focus Optics. These mirrors were chosen because they were extremely efficient and allowed for the highest possible pulse energy to be delivered to the fluorescent sample and to the SFG crystal. However, it was found that the multi-stack dielectrics introduced significant phase distortion for pulses shorter than ~50 fs. We suspect that the phase distortion originated from frequency dependent interference of the reflections from the different

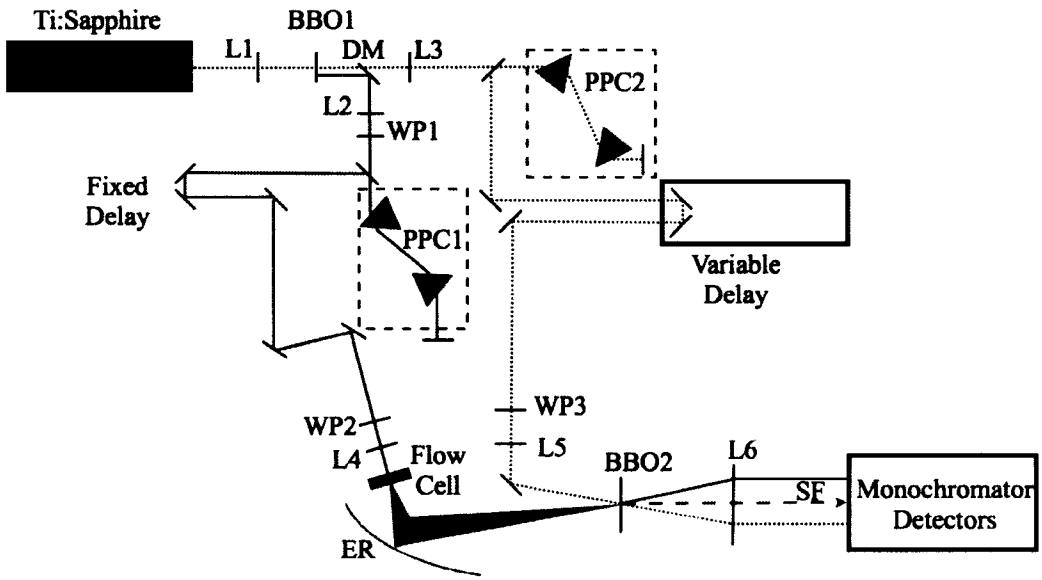


Figure 2.1 Schematic of the broadband fluorescence upconversion instrument used in this work.

dielectric layers, or stacks, that are the source of the mirrors high efficiency. The phase distortion was observed in the autocorrelation traces of oscillator pulses after reflection off of one of these multi-stack mirrors. The autocorrelation of the pulse before reflection

from the multi-stack mirror produced a single peak with almost no shoulder. The autocorrelation trace of the pulse after reflection from the mirror surface was yielded three separate peaks. Reflection off a multiple multi-stack mirror led to autocorrelation traces with multiple peaks. The phase distortion problem was alleviated by replacing the New Focus mirrors with CVI Laser Corp. TML model single stack dielectric mirrors. The TLM model mirrors are specifically designed for ultra-short/broadband laser pulses and do not distort the pulse. This was verified by observing the autocorrelation trace before and after reflection off the single stack mirrors. The only difference between the observed trace before and after reflection was a slight increase in temporal width of approximately 2-5%. According to the manufacturer, this increase is to be expected.

Another performance issue that arose due to the broadband pulses used for this experiment was that of group velocity dispersion (GVD) introduced by transmissive optics [25]. Because the speed of light in optical media is frequency dependent, one portion of a broadband pulse will be retarded with respect to another portion of the pulse with another frequency. The effect of this differential retardation is that pulses of sufficient bandwidth will be stretched in time. This phenomenon is called chirp[25]. For most optical media the speed of light decreases with increasing frequency. Because lower frequencies will be retarded less than higher frequencies, the instantaneous frequency of the pulse with respect to an observer in the laboratory frame of reference appears to shift to higher frequency.

In the instrument described here, the GVD was compensated for by prism compression (see PPC1 and PPC2). The prisms geometrically disperse the pulse as a function of wavelength, thus as indirectly as a function of frequency. Utilizing this

geometric dispersion, the lower frequency portion of the pulse can be made to travel a greater distance than the higher frequency portion. This difference in travel distance of the high and low frequency portions of the pulse compensates for the chirp introduced by the transmissive optics[25].

The prism compressors compensate for the accumulated GVD due to optics preceding them, but they can also pre-compensate for GVD due to optics that follow. Since the transmissive optics used in this instrument impose an up-chirp (where the instantaneous frequency within the pulse increases as a function of time) the prism compressor can be configured to produce a down-chirp on exiting pulses. The amount of applied down-chirp can be adjusted by varying the distance between the prisms. Larger prism separation results in a larger amount of applied down-chirp[25].

Because the pump and gate pulses have very different wavelengths that result in very different dispersion characteristics, two prism compressors were necessary to compensate for the accumulated GVD of each beam path. The distance between the prisms of each pair was adjusted separately. First, the gate pulse prism pair distance was adjusted by starting with a separation of approximately one meter. Then the gate pulse was measured by intensity auto-correlation at a point just before the SFG mixing crystal, see Figure 2.2. Next, the prism separation was increased slightly and auto-correlation trace was measured. This was repeated until the width of the autocorrelation did not decrease.

The pump pulse could not be measured by with the same auto-correlation instrument (Femtochrome, model FR-103hs) because the auto-correlator optical elements are optimized for 600-900 nm light. Instead of a simple intensity auto-correlation, the

pump pulse width was measured by recording a cross-correlation FROG trace with the same experimental configuration used to record FROG traces of solvated dye fluorescence shown in Figure 2.1. At each delay stage position a spectrum of the cross-correlation SFG signal was collected and integrated to give a total integrated intensity vs. time. The prism separation was the increased and another wavelength integrated cross-correlation was recorded. The prism separation was incrementally increased until the width of the wavelength integrated cross-correlation trace no longer decreased.

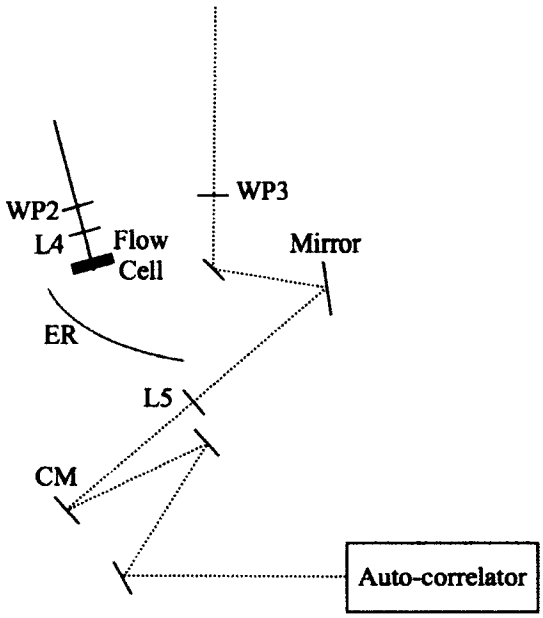


Figure 2.2 Schematic of the experimental configuration for measuring the auto-correlation trace of the gate pulse.

Data Acquisition

Initially, the SFG signal was optimized by setting the variable delay to approximately 30 ps after the zero delay position and observing the signal strength using a photomultiplier tube (PMT) with photon counter. The ellipsoidal reflector (ER) position, SFG crystal (BBO2) angle, collimation lens (L6) position and monochromator wavelength were optimized to give the highest reading on the PMT/photon counter detection.

Next, the monochromator was converted to CCD detection by removing the turning mirror required for detection by the PMT. The SFG signal was centered on the active area of the CCD using the CCD readout within the data acquisition software. The software allows for selection of a sub-region of the CCD. In a procedure referred to as binning, the CCD hardware summed the intensity from four adjacent pixels of the CCD and stored the result as a single valued pixel. The binned pixels from the sub-region were averaged along the vertical dimension and recorded by the computer along with the position of the variable delay stage.

The effect of binning the 512 x 1024 CCD active region on the effective resolution of the instrument is negligible. The resulting binned image has a resolution of 0.54 nm/pixel. A simple calculation shows that determination of the wavelength of a signal with error less than 0.54 nm in the region of 300 nm requires a measurement lasting at least 600 fs. Because most of the dynamics we wish to measure occur on time scales shorter than 500 fs, the operating resolution of the CCD after binning is more than sufficient given the amount of detail expected to be observed with fast dynamics. Typically, time resolved fluorescence (TRF) spectra were collected by selecting a vertical

sub-region of the CCD that included the most intense portion of the upconverted spectrum plus 3-4 rows above and below. The signal (solid concentric ovals) and sub-region (dashed rectangle) are depicted in Figure 2.3 and Figure 2.4. This resulted in recording a region of the CCD that was 256 binned pixels wide and 8-12 binned pixels high depicted in Figure 2.4. The rows of this region were then averaged to produce a single TRF spectrum. The CCD sub-region was exposed for 20 seconds at each delay stage position. The CCD was held at -80°C with a liquid nitrogen-aided Peltier cooler to reduce thermal noise.

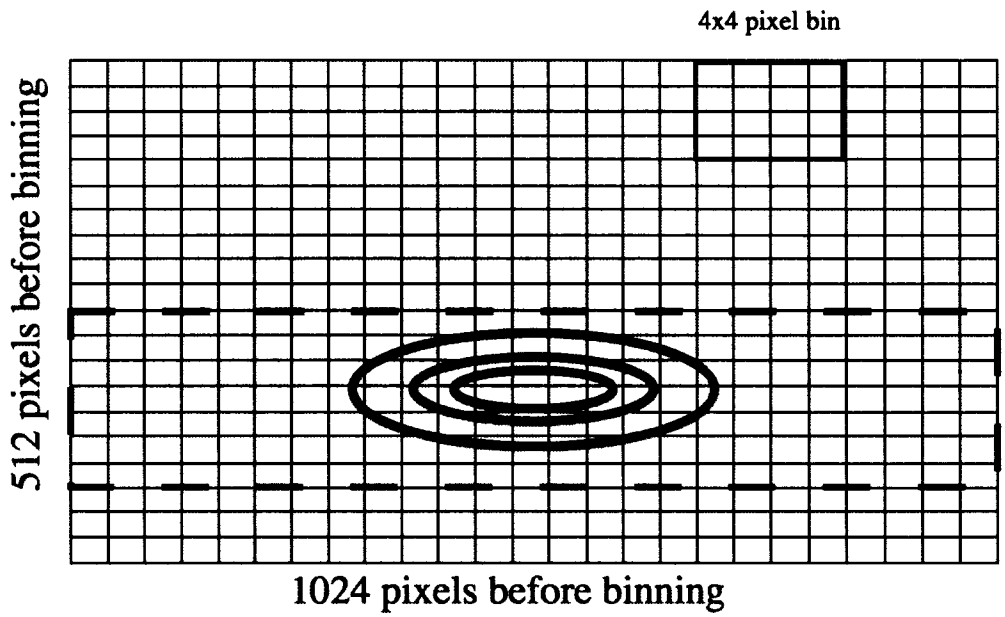


Figure 2.3 Full resolution map of the CCD before binning. The ovals depict the upconversion signal. The solid rectangle depicts the pixel region that is to be binned.

To further minimize the background, the room lights were completely extinguished.

Several background spectra were obtained by recording the CCD signal approximately 1 ps before the start of the fluorescence signal. A typical background spectrum is shown in Figure 2.5. The intensity of the background in the SFG region (300-360 nm) is usually 600-800 counts.

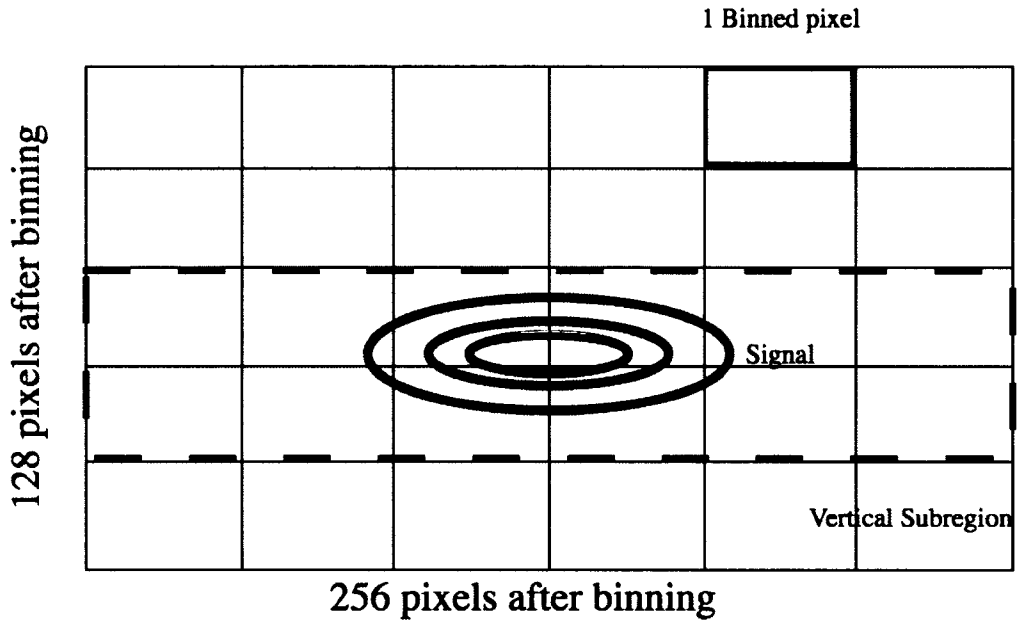


Figure 2.4 The region in the dashed oval represents the vertical sub-region selected by the user.

This background spectrum should account for background scatter from all sources not gated by the gate pulse. These sources include non-upconverted fluorescence in the visible range, the gate pulse centered about 800 nm, and residual pump pulse light around 400 nm that are not completely eliminated by the short pass filter on the monochromator.

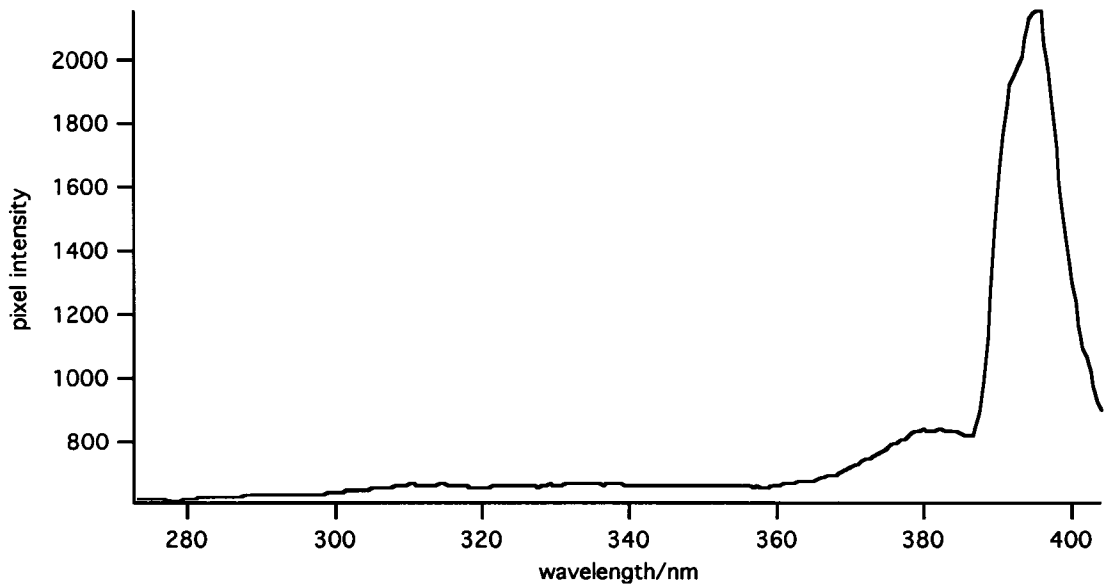


Figure 2.5 A typical background spectrum collected approximately 1 ps before the start of the gated fluorescence signal.

The step size of the variable delay was set to an appropriate value for the predicted time scale of the dynamics to be observed and for the total time allotted for the experiment. For instance, solvation dynamics in small polar solvents such as water, acetonitrile, and methanol proceed on timescales of approximately 1-2 ps with some processes occurring on time scales of approximately 10-50 ps [10]. To measure the shorter time dynamics occurring on the 1-2 ps timescale a small time step such as 33.36 fs was chosen in order to give the most detailed information on the rapid dynamics. Longer time dynamics occurring on a 10-50 ps timescale would not require such small steps, so larger steps such as 100 fs or larger could be used. Trials lasting between 1-2 hours were desired to minimize the effect of ambient temperature changes within the time spanned by a single experiment. Each spectrum took 20 seconds to acquire and the delay stage could take up to 20 seconds to move for large step sizes of 50 ps. Therefore, in one hour 90-160 spectra could typically be collected. At least two trials of each experiment

were performed.

The experiment was controlled and TRF spectra recorded by software written in Labview. The Labview software performed two major functions. First, the software controlled the CCD hardware, manipulated the recorded image and wrote the data to disk. The binned image was transferred from the CCD interface hardware to the Labview software where the row averaging was performed. The software performed the row averaging by adding the intensity of each of the pixels in a column of the user selected sub-region and then divided each column sum by the number of rows in the sub-region producing an averaged row. Then each value of the averaged row was written as a single precision floating point number to the hard disk of the computer. Next, the Labview software instructed the CCD to clear the accumulated charge from the transfer registers and wait for the command to accumulate the next image.

The second major function of the software was to control the variable delay stage. The software transmitted the base step size, either a half or whole step, and the trigger signal to move the stage. The half step size corresponded to a time delay of 16.68 fs of the gate pulse with respect to the pump pulse. Consequently, all of the time intervals of the TRF spectra were multiples of the base step size of 16.68 fs. The intervals were controlled by varying the number of trigger pulses sent to the variable delay stage. The running position of the delay stage was saved with the corresponding spectrum.

The data acquisition process has been summarized in a flow chart (Figure 2.6). Triangles pointing downward represent commands sent from the computer to the various hardware components. The triangle pointing upward represents data received by the computer. Solid rectangles represent image processing performed outboard, by the CCD

controller, while ovals represent image processing done internally by the computer. Each loop of the data acquisition program started with stepping the delay motor and ended with writing the spectrum to the data file on the hard disk.

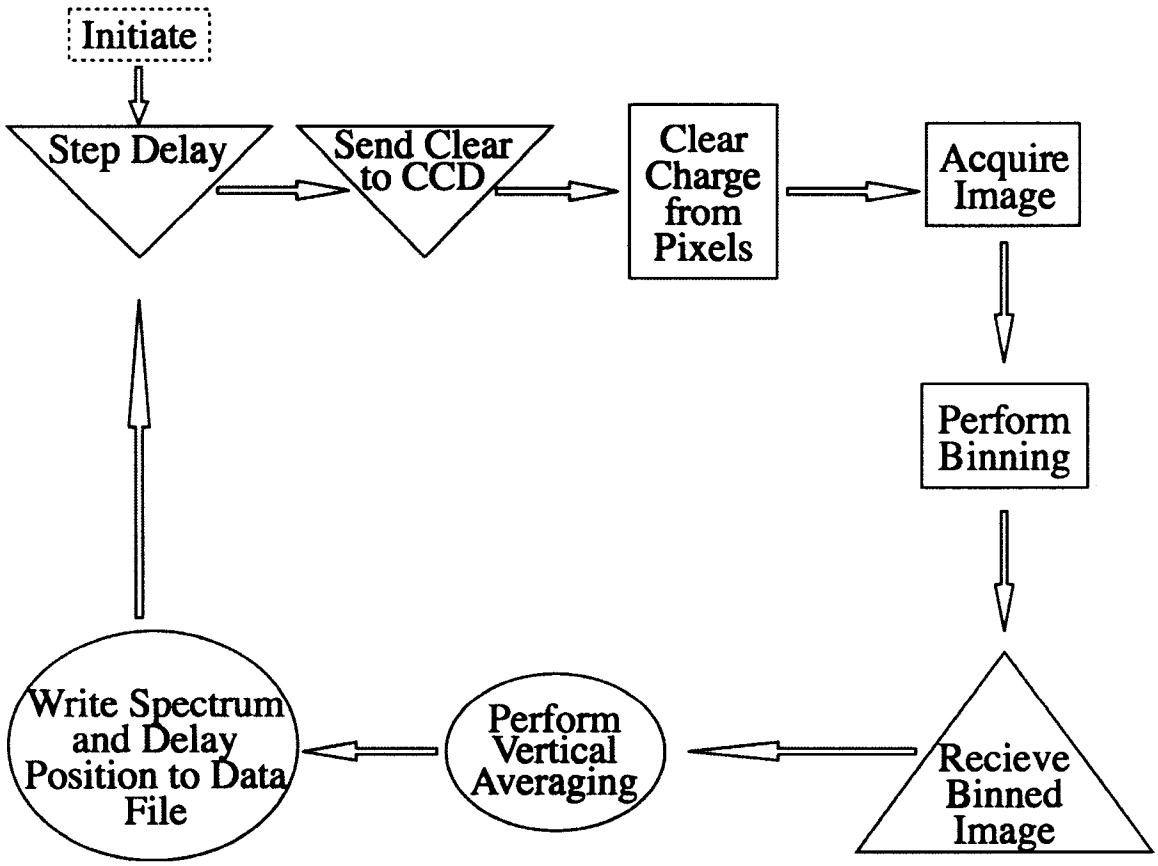


Figure 2.6 Flow chart of the basic steps of the data acquisition process performed by the control computer and the CCD hardware. Triangles pointing downward represent commands sent from the computer to the various hardware components, the triangle pointing upward represents data received by the computer, rectangles represent image processing performed outboard, ovals represent image processing done internally by the computer.

First, the computer sends a signal to the delay stepper motor to move to the first delay position. Then the computer sends a clear charge signal to the CCD. The CCD then clears the charge from the pixels and the transfer registers. Next, the image is acquired on

the CCD and binning is performed within the camera hardware. Then the binned image is sent to the computer where the vertical, or row, averaging is performed to generate a spectrum from the binned image. The last step in the acquisition process is to write the spectrum and the delay stage position to the data file on the hard disk.

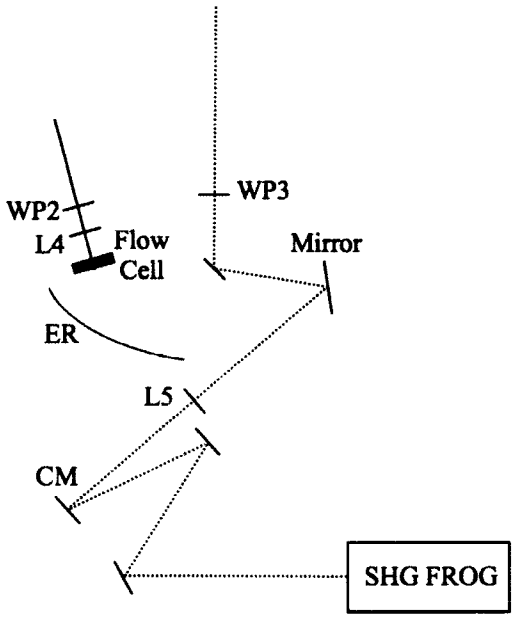


Figure 2.7 Schematic of the modified layout used to measure the SHG-FROG of the gate pulse.

Two other diagnostic measurements were performed for each experiment. First, the electric field of the gate pulse was obtained by measuring its FROG trace just before BBO2 (see Figure 2.7). This was accomplished by removing L5 and replacing the BBO2 with a mirror to reflect a collimated beam away from the experiment. In order to replicate its dispersive effects on the gate pulses, L5 was placed back in the beam path as shown in

the Figure 2.7. A curved mirror was then employed to collimate the beam after being focused by L5. This collimated beam was then directed into the SHG FROG instrument and the electric field was obtained from the FROG software.

The second diagnostic performed was to measure the cross-correlation SFG-FROG trace of the pump pules. This was accomplished using the same configuration used for recording the TRF. The SHG crystal was tuned to an optimal angle for mixing the gate and pump pulses and then pumping pure solvent (without dye) through the flow cell. The electric field was then calculated from the recorded FROG trace using custom-built phase retrieval software.

Chapter 3

Data Analysis Methods and Results

Introduction to Broadband Time Resolved Fluorescence Data Analysis Techniques

In contrast to the traditional narrow bandwidth upconversion method, time and frequency information is collected simultaneously in the broadband fluorescence upconversion. Figure 3.1 (page 33) shows the upconverted time resolved fluorescence spectra of Coumarin 153 acetonitrile. This figure shows significantly higher frequency resolution than is normally collected from the tradition upconversion experiment. The upconverted fluorescence signal peaks at approximately 307 nm and shifts to the red during the first picosecond of signal. After the first picosecond the peak wavelength of the upconverted fluorescence remains relatively constant. The region of higher intensity on the lower wavelength edge of the spectrograph is the upconversion, or sum frequency, signal of the residual pump pulse centered at 404 nm with the gate pulse centered at 808 nm.

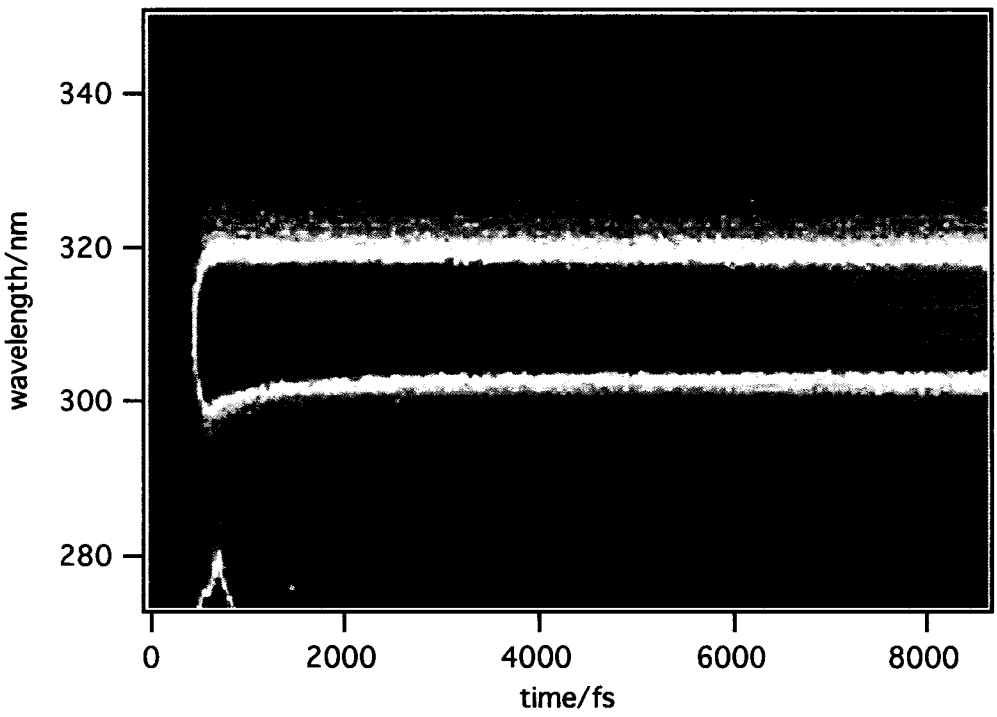


Figure 3.1 A color image of the time-resolved fluorescence from Coumarin 153 in acetonitrile collected using sum frequency gating.

Collecting both time and frequency data at such a high density provides built-in data redundancy because the time and frequency data are not completely independent. The Fourier transform relation between bandwidth and time requires that an event with short temporal dynamics have a large bandwidth. When both the frequency and time sampling resolution is high, as they are for data collected with this instrument, there is data redundancy because fast temporal dynamics cannot contain arbitrarily complex spectral features that evolve at the same rate. Consequently, frequency information is recorded at resolutions higher than is necessary to resolve dynamical spectral features. The phase retrieval analysis takes advantage of this redundancy. Furthermore, when performing more traditional analysis such as peak fitting of each time resolved upconverted spectrum, substantially higher frequency resolution results in better

lineshape fits, higher accuracy in peak frequency determination and better statistical measures compared with collecting ten to twelve individual fluorescence decays one-at-a-time.

Data Analysis

Two approaches to analyzing the time-frequency data were employed. First, the gated fluorescence spectra were directly fit to a sum of Gaussians, a method similar to the traditional method of spectral reconstruction from individual fluorescence decays.[2, 7] The second method was based on phase retrieval algorithms developed for characterization of ultrafast laser pulses.[30] These algorithms attempt to reconstruct the electric field of the fluorescence pulse from its nonlinear optical interaction with an independently characterized reference pulse.

The aim of both analytic techniques was to track the free energy change of the dye/solvent system through the frequency shift of the fluorescent dye. The configuration of the solvent about the dye influences the frequency of the dye fluorescence. Thus, the solvent's reaction to the dye decreases the free energy of the system, which caused a time dependent frequency shift of the fluorescence.

General Preparation of Data for Analysis

The collected spectra were processed prior to analysis for dynamical information. The processing and analysis was performed in the graphing and analysis software package Igor Pro. First, the data were loaded into Igor Pro with custom built functions to read the delay stage position and then the corresponding spectral data. Within Igor Pro, the time resolved fluorescence data were stored as a 2 dimensional matrix with each time

resolved spectrum occupying a column in the matrix. This matrix was then plotted as a 2-D time-frequency data matrix (TFDM) with time running along the x-axis and wavelength along the y-axis. The intensity of each time and spectrally resolved point was represented as a different color or shade of gray. This type of representation is shown in Figure 3.1, at the beginning of this section.

One source of noise in a recorded time resolved fluorescence spectrum is cosmic rays striking the CCD detector. This is a well-characterized effect is pervasive in all experiments utilizing highly sensitive CCD detectors with integration times of several seconds or longer.[31] Highly energetic particles made up of mostly protons and alpha particles, strike the CCD detector at random locations and random intervals. This impact creates an electron-hole pair, which is recorded in the charge collection layer of the pixel. These events result in drastically higher charge counts confined to an area of one to approximately five pixels. Since the event of the a cosmic ray impact is random and the resulting pixel value is up to two orders of magnitude higher than the surrounding pixels, the resulting spike in the time resolved spectrum can easily be identified and removed. Spikes were removed from the time-frequency data matrix by a custom built function in Igor Pro. This function parsed a user-defined area of the TFDM and replaced the most intense time-frequency point with the average of its 24 nearest neighbors in the 2-D matrix.

Next, there was a substantial, but constant background signal that had to be subtracted from the actual upconverted spectra. Several background spectra were collected up to 1 ps before the start of the fluorescence signal in order to ensure that none of the residual pump pulse was present in the background. Four to six of these

background spectra were averaged to reduce the effect of random pixel fluctuations on the spectrum caused by stray light and thermal noise. Then, this averaged background spectrum was subtracted from all of the measured time resolved fluorescence spectra from a single experiment trial. A new background spectrum was calculated for each trial of each data set. This was done to minimize the effects of changes in ambient light and thermal load on the CCD.

Method One: Direct Fitting to a Sum of Gaussians

Details of the Sum of Gaussians Fitting Analysis

After spike removal and background subtraction, the time resolved fluorescence spectra were analyzed for dynamical content by fitting the upconverted spectrum at each delay stage position to a sum of two Gaussian functions in Igor Pro. There were seven adjustable parameters for the fit according to the following equation,

$$I(\nu;t) = y_t + A_{1,t}Exp\left\{-\left(\frac{\nu-\nu_{1,t}}{width_{1,t}}\right)^2\right\} + A_{2,t}Exp\left\{-\left(\frac{\nu-\nu_{2,t}}{width_{2,t}}\right)^2\right\}$$

Equation 3.1

where y_t is the baseline offset, $A_{i,t}$ is the amplitude, $\nu_{i,t}$ is the peak frequency, and $width_{i,t}$ is the spectral width for each of the two Gaussians.

Initially the data recorded from the CCD was sampled linearly in wavelength. An appropriate frequency scale was generated from the wavelength scale output by the upconversion instrument by taking the inverse of the wavelength and multiplying by the speed of light. This frequency scale was used as the independent variable in the Gaussian fit function.

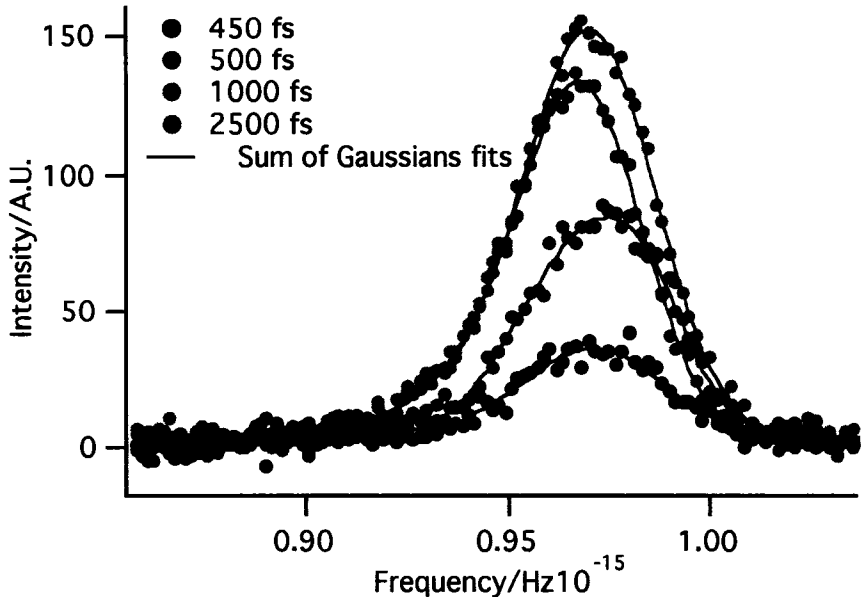


Figure 3.2 Time resolved spectra of Coumarin 153 in acetonitrile at a range of delay times following excitation; SFG data are given by points while fits to **Equation 3.1** are given by red lines.

The first time resolved fluorescence spectrum, whose maximum peak intensity was at least twice that of the RMS noise floor of the data set, was used as a template to define the relevant spectral region. Typically each spectrum had an RMS noise floor of 10-15 counts so the first time resolved fluorescence spectrum usually had a maximum intensity of at least 20 counts. This spectrum was fit using Igor Pro's built-in Levenberg-Marquardt algorithm where the initial guesses for the fit coefficients were supplied by the user. The resulting parameters from the fit of the first spectrum were used as the initial guess for the next spectrum in the sequence. Likewise, the initial guesses for the fit parameters of the subsequent upconverted fluorescence spectra were the results of the fit of the spectrum from the previous time step. All fit parameters as well as their standard deviations and the total chi-squared value were stored for each fit produced for each time

resolved spectrum. Fits to equation 1 for several time resolved fluorescence spectra of C153 in acetonitrile at crystal angle 59° are shown in Figure 3.2.

The average frequency for each time resolved spectrum was calculated by integrating the intensity-weighted frequency and then normalizing by the total integrated intensity obtained from the fits according to Equation 3.2.

$$\bar{\nu}(t) = \frac{\int I(\nu;t)\nu d\nu}{\int I(\nu;t)d\nu}$$

Equation 3.2

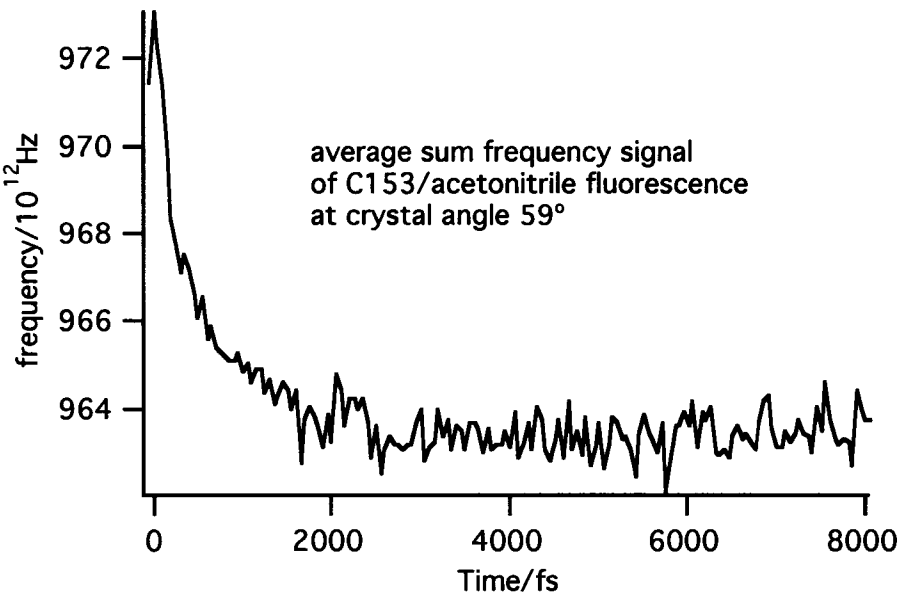


Figure 3.3 Time resolved average frequency of the upconverted fluorescence signal from C153/acetonitrile calculated using Equation 3.2.

Substitution for $I(\nu;t)$ gives

$$\bar{\nu}(t) = \frac{\int \left(A_{1,t} Exp\left\{ -\left(\frac{\nu - \nu_{1,t}}{width_{1,t}} \right)^2 \right\} + A_{2,t} Exp\left\{ -\left(\frac{\nu - \nu_{2,t}}{width_{2,t}} \right)^2 \right\} \right) \nu d\nu}{\int A_{1,t} Exp\left\{ -\left(\frac{\nu - \nu_{1,t}}{width_{1,t}} \right)^2 \right\} + A_{2,t} Exp\left\{ -\left(\frac{\nu - \nu_{2,t}}{width_{2,t}} \right)^2 \right\} d\nu}$$

Equation 3.3

The baseline offset, whose magnitude was typically 1-2% that of the peak intensity, was omitted from this average frequency calculation because the offset was constant across the entire spectrum and therefore contributed no spectral information and would not affect the average frequency. Figure 3.3 shows the plot of $\bar{\nu}(t)$ C153 in acetonitrile at crystal angle 59°.

The average frequencies (from Equation 3.4) or peak frequencies from each time resolved spectrum were used to calculate a solvent response function. The solvent response function for time resolved fluorescence is defined as

$$C(t) = \frac{\bar{\nu}(0) - \bar{\nu}(t)}{\bar{\nu}(0) - \bar{\nu}(\infty)}$$

Equation 3.4

where $\bar{\nu}(t)$ is given in Equation 3.2 (or can be the peak frequency) and $\bar{\nu}(\infty)$ corresponds to the last point calculated from the data set. The *C(t)* function is fitted to a bi-exponential decay,

$$y_0 + A_1 Exp\left(-\frac{x - x_0}{\tau_1} \right) + A_2 Exp\left(-\frac{x - x_0}{\tau_2} \right)$$

Equation 3.5

where A_i is the amplitude of each component, y_0 is the baseline offset, x is the time axis, with x_0 the time offset, and τ_1 and τ_2 are the characteristic time constants for the decays.

Figure 3.4 displays plots of the solvent response function for C153 in acetonitrile at three crystal angles along with bi-exponential fits to the data. The left panel displays solvent response function calculated using the average frequency (Equation 3.2). The right panel shows the response function calculated using the peak frequency given by the higher frequency Gaussian from the sum of Gaussians fitting function. The fit parameters for the solvent response function calculated from the average frequency are tabulated in Table 3.1. The fit parameters for the peak frequency solvent response function are tabulated in Table 3.2. In each table of fit parameters the error figures were produced by the fitting algorithm and represent the error for a 95% confidence interval. The numbers in parentheses are the number of trials averaged to produce the solvent response function.

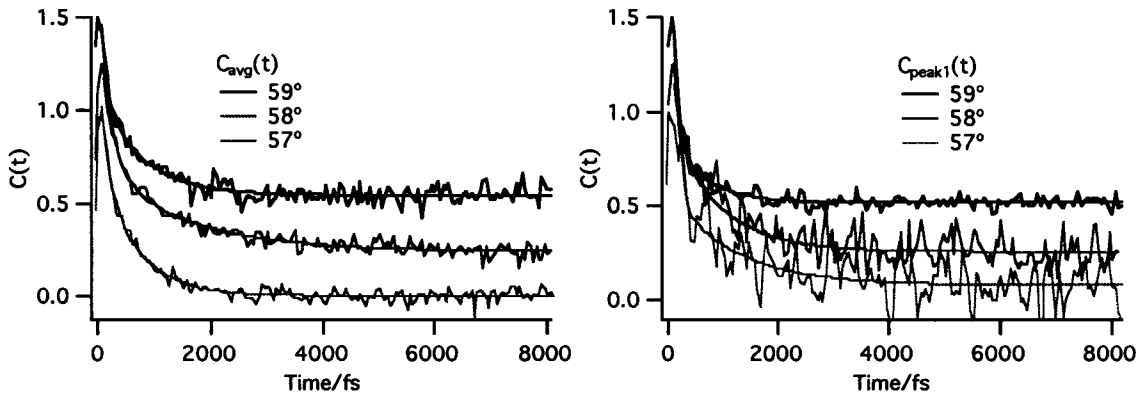


Figure 3.4 (Left) Solvent response functions at three different nonlinear optical crystals calculated from the time resolved average frequency. (Right) Solvent response functions calculated from the time resolved peak frequency.

	59° (*n*=1)	58° (*n*=2)	57° (*n*=3)	Lit.1
A_1	0.43 ± 0.08	0.53 ± 0.04	0.35 ± 0.08	0.686
τ_1(fs)	110 ± 36	207 ± 32	140 ± 43	89
A_2	0.57 ± 0.08	0.47 ± 0.04	0.65 ± 0.08	0.314
τ_2(fs)	736 ± 90	1525 ± 160	646 ± 59	630

Table 3.1 Tabulated bi-exponential fit parameters of C153/acetonitrile $C_{avg}(t)$ for three crystal angles and for one literature source.[10] The numbers in parentheses are the number of trials (*n*) averaged to produce each $C_{avg}(t)$.

	59°(*n*=1)	58°(*n*=2)	57°(*n*=3)
A_1	0.66 ± 0.05	0.56 ± 0.1	0.75 ± 0.13
τ_1(fs)	78 ± 11	119 ± 42	410 ± 145
A_2	0.33 ± 0.05	0.44 ± 0.04	0.25 ± 1.3
τ_2(fs)*	613 ± 80	2394 ± 616	12.7 ± 105 ps*

Table 3.2 Tabulated bi-exponential fit parameters of C153/acetonitrile $C_{peak1}(t)$ for three crystal angles.

By contrast, Figure 3.5 depicts the solvent response function of methanol calculated from average frequencies for three crystal angles offset by 0.25 to separate them on the plot. The tri-exponential fit parameters are tabulated below in Table 3.3. These response functions exhibit the longer time dynamics of methanol solvent relaxation.

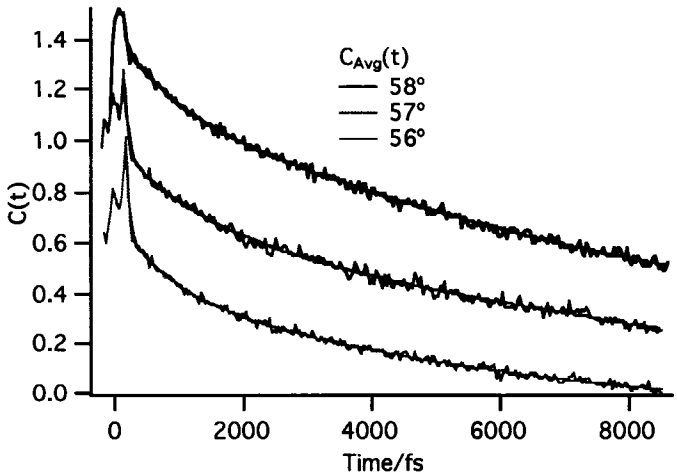


Figure 3.5 Solvent response function of C153 in methanol at three crystal angles with tri-exponential fits.

	58° (*n*=2)	57° (*n*=2)	56° (*n*=4)	Lit.1
A_1	0.08 ± 0.01	0.18 ± 0.01	0.33 ± 0.01	0.101
τ_1(fs)	36 ± 13	41 ± 7	49 ± 5	30
A_2	0.15 ± 0.01	0.18 ± 0.05	0.18 ± 0.02	0.340
τ_2(fs)	739 ± 132	1196 ± 333	791 ± 149	280
A_3	0.77 ± 0.05	0.64 ± 0.23	0.49 ± 0.02	0.298

τ_3(fs)	9181 ± 1382	12976 ± 9619	6768 ± 1369	3200
A_4	N/A	N/A	N/A	0.261
τ_4(fs)	N/A	N/A	N/A	15300

Table 3.3 Tabulated tri-exponential fit parameters of C153/methanol $C_{avg}(t)$ for three crystal angles.

Fitting to a sum of Gaussians results in two peak frequency parameters that can independently track spectral diffusion in different portions of the time resolved fluorescence. The separation between these two peak frequencies can be calculated at each time point to determine if the degree of decoupled spectral diffusion. Figure 3.7 shows the calculated peak differences. The error bars in the right panel are the sample deviations of at each point for the two angles (57° and 58°) that have more than one trial.

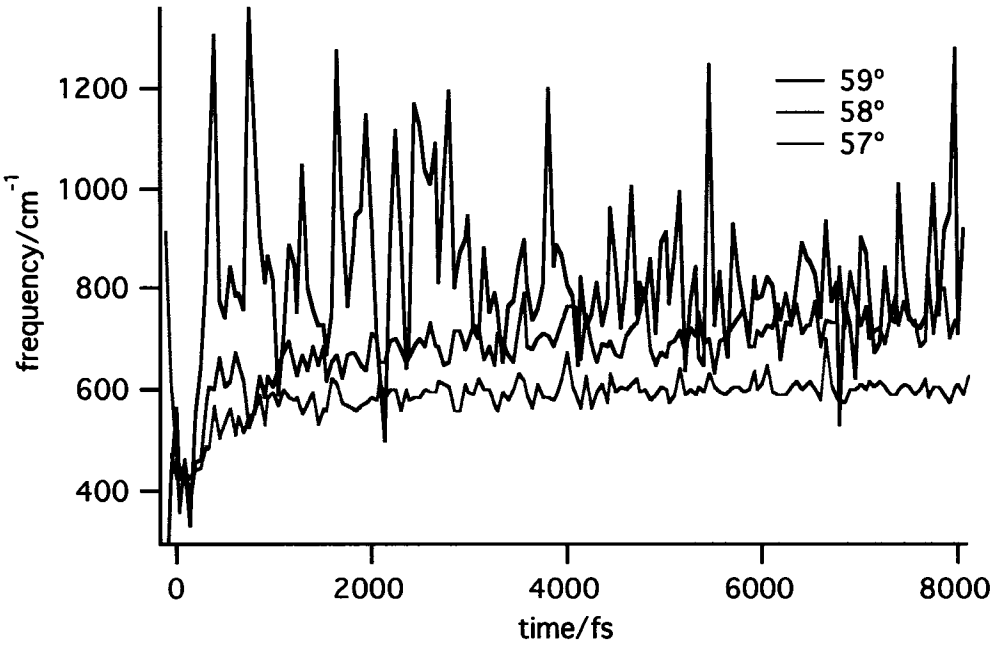


Figure 3.6 Peak frequency difference calculated from the sum of Gaussians fit parameters for three angles for the full data range.

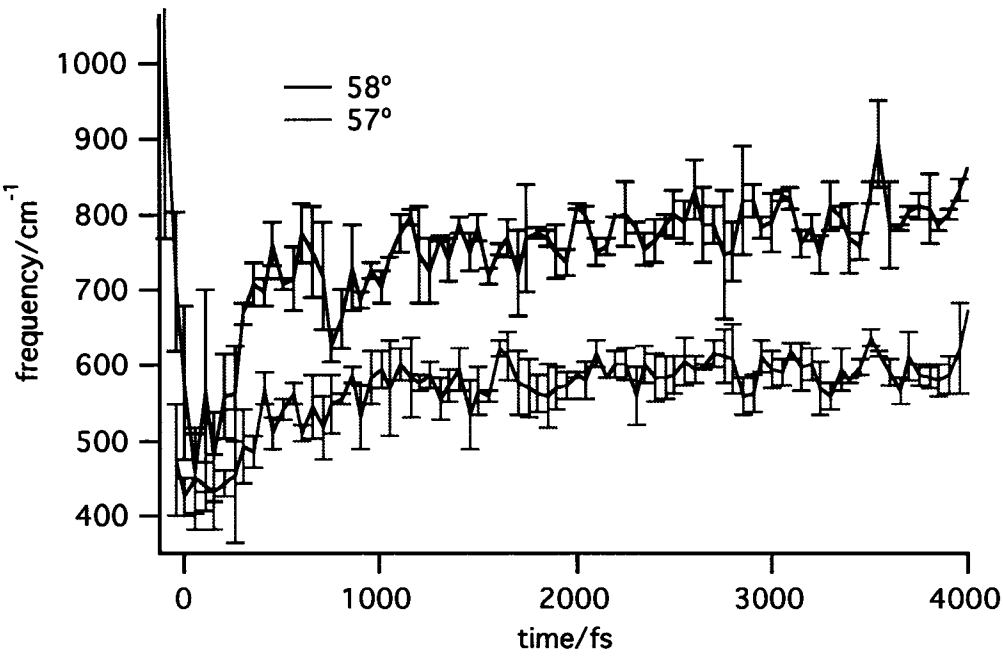


Figure 3.7 Peak frequency difference showing 58° and 57° with error bars over the first 4000 fs where the 58° trace has been shifted upward by 100 cm^{-1}.

Multiple trials of the same experiment reveal details of the early time dynamics. Figure 3.8 shows the *C(t)* for C153/acetonitrile fluorescence at crystal angles 59°, 58° and 57°, where the 57° curve has been shifted downward by 0.2 and the 59° curve has been shifted downward by 0.4. The error bars are calculated from the sample deviations for each time point. There appear to be features in all three curves at approximately 250 and 500 that are larger than the error bars associated with them. Additionally, in there appears to be a feature at 750 fs in the 58° curve.

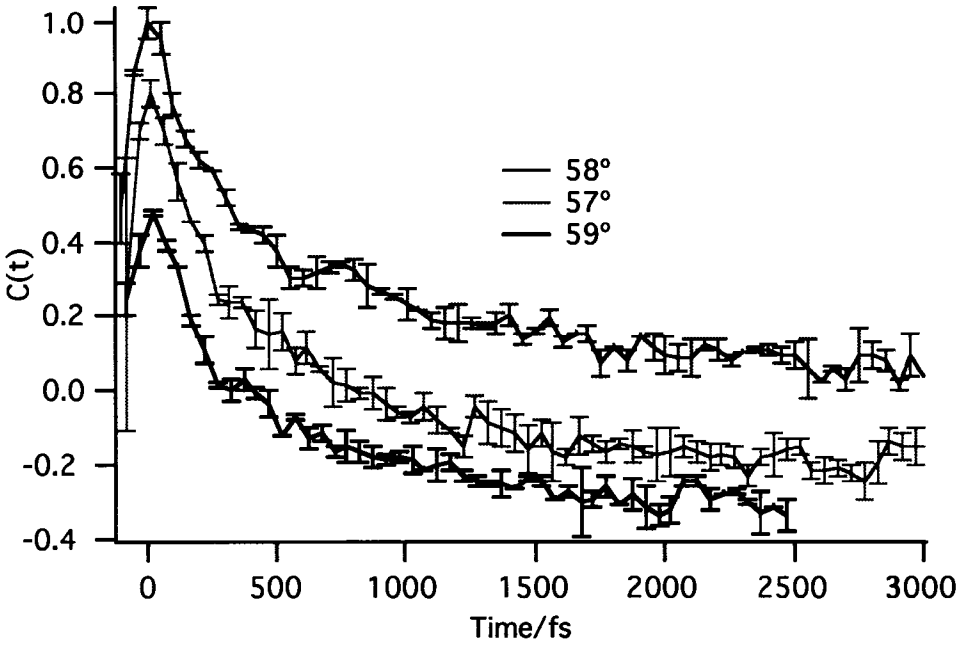


Figure 3.8 Close up of *C(t)* for C153/acetonitrile fluorescence at crystal angles 59°, 58° and 57° showing just the first 3000 fs of data.

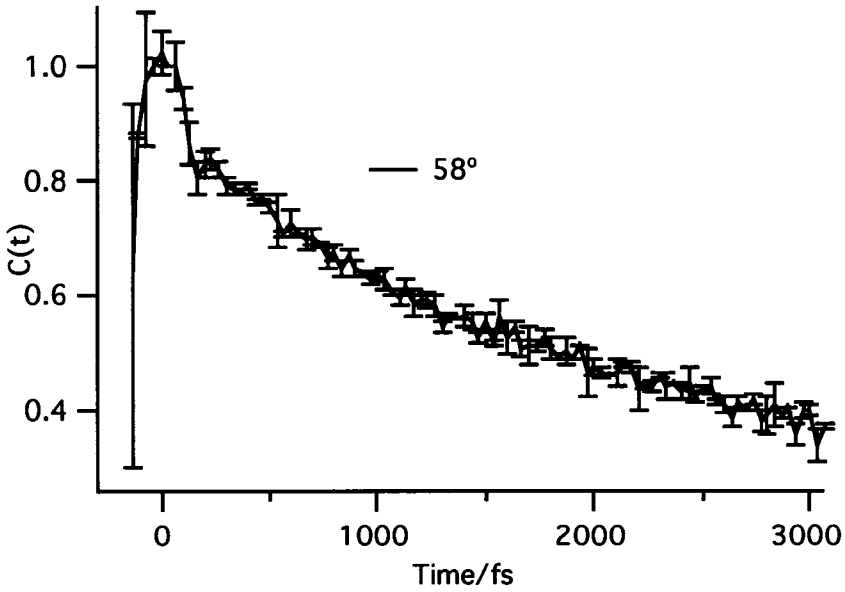


Figure 3.9 Close up of *C(t)* for C153/methanol fluorescence at crystal angle 58°.

Method Two: Data Analysis by Phase Retrieval Algorithm

The fine details of the solvation dynamics revealed by the results shown in Figure 3.8 suggest dynamics that occur on timescales on the order of 100 fs or less. In this regime, it would be useful to have a robust method for deconvolving the influence of the tempo-spectral characteristics of the gate pulse used in the upconversion process. The aim of phase retrieval is to accomplish this deconvolution.

Because of the particular mathematical manipulations involved in retrieving the phase of an optical signal, more preparation of the data was needed. After spike removal and background subtraction, the time resolved fluorescence spectra were re-sampled to a grid that was linear in frequency as well as time. In the phase retrieval algorithm several one-dimensional fast Fourier transforms were calculated. The calculation of the FFTs required that the frequency point spacing be the inverse of the total time window and the time point spacing be the inverse of the total bandwidth. Typically the bandwidth of the upconversion signal together with number of grid points allowed by physical limitations on calculation time and computer memory restricted the time window to less than 5 ps. The resampling was performed using a built-in image spline-interpolation function in Igor Pro. The electric field of the gate pulse was also resampled to the same time point spacing.

Next, the re-sampled FROG trace and gate pulse electric fields were written to a single computer file along with parameters to control the phase retrieval software. This file was then transferred to an Xserve (Apple Computer) for execution of the phase retrieval software.

The output of the XFROG modified phase retrieval algorithm is an approximation to the temporal electric field of the fluorescence nominally deconvolved from the gate pulse. From this temporal field, the temporal phase and temporal magnitude can be calculated. The temporal phase is useful because it provides a succinct measure of how the carrier frequency is modulated over the duration of the optical pulse. The Fourier transform of the electric field can also be calculated to produce the power spectrum. Figure 3.10 shows the re-sampled XFROG data from C153/acetonitrile at a crystal angle of 59°, which was supplied to the phase retrieval algorithm.

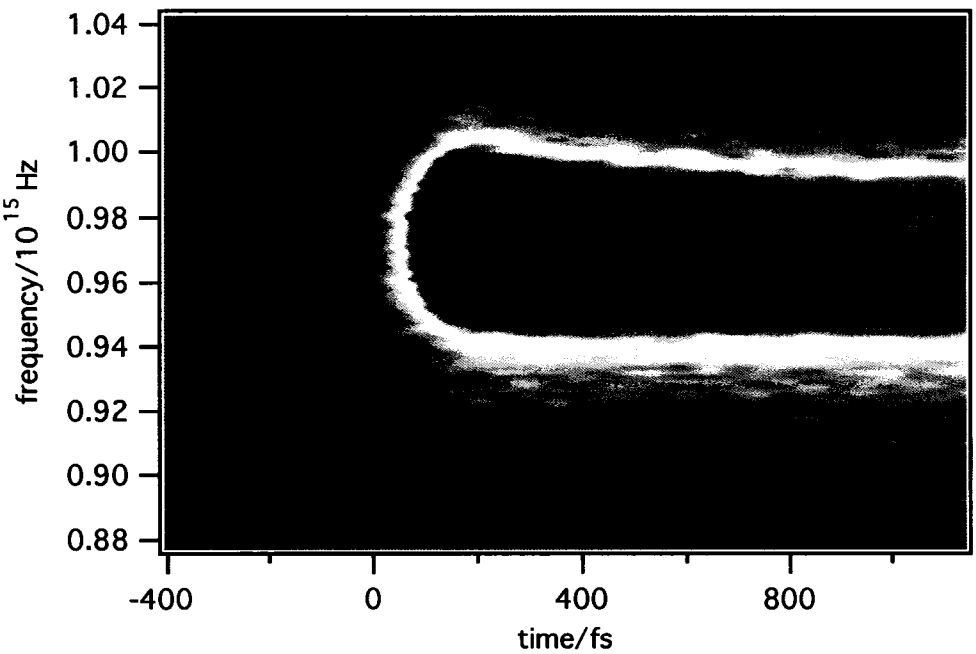


Figure 3.10 Re-sampled cross correlation frequency resolved optically gated fluorescence from C153 in acetonitrile at crystal angle 59°.

Figure 3.11, below shows the retrieved electric field of the gate pulse from the commercial SHG-FROG instrument.

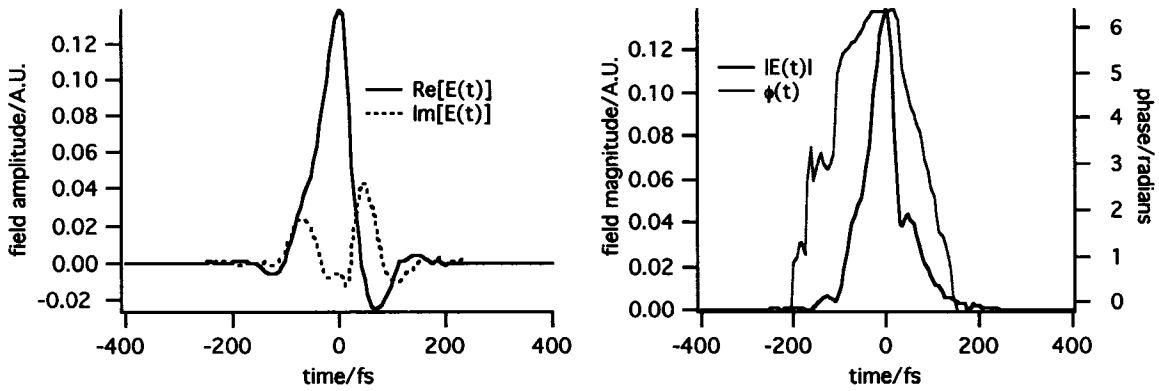


Figure 3.11 The independently measured complex electric field of the gate pulse. Left panel: complex representation. Right panel: magnitude and phase representation.

The output of the phase retrieval algorithm is the complex electric field $E(t)$. The phase $\phi(t)$ and magnitude $A(t)$ are calculated to more easily visualize the pulse and its spectral characteristics.

$$\phi(t) = -\operatorname{Arctan}\left\{\frac{Im[E(t)]}{Re[E(t)]}\right\}$$

Equation 3.6

$$A(t) = |E(t)|$$

Equation 3.7

An instantaneous frequency can be defined as

$$\omega(t) = \omega_0 + \frac{d\phi(t)}{dt}$$

Equation 3.8

where ω_0 is the carrier frequency. The change in instantaneous frequency is

$$\frac{d\omega(t)}{dt} = \frac{d^2\phi(t)}{dt^2}$$

Equation 3.9

Figure 3.12 shows the complex representation of the output of the phase retrieval algorithm for C153/acetonitrile fluorescence.

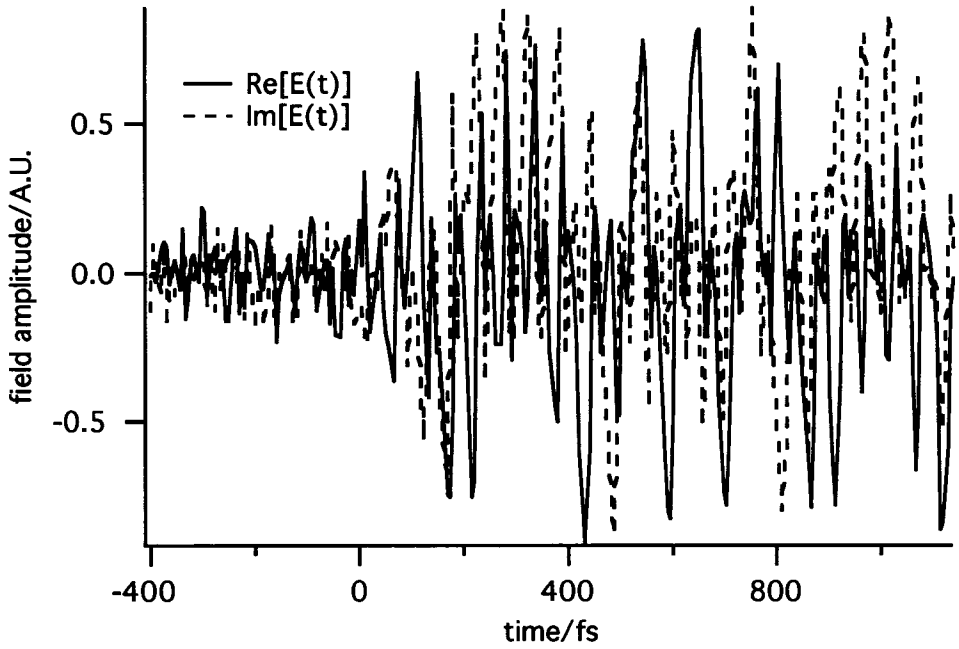


Figure 3.12 Complex electric field $E(t)$ returned from the phase retrieval algorithm of the fluorescence signal from C153 in acetonitrile at crystal angle 59°.

In order to apply the results from the phase retrieval algorithm, we must establish a connection to the change in the free energy of the solute/solvent system. Recall that Equation 3.8 relates the time derivative of the phase to the instantaneous frequency. Because of the solvatochromic shift of the solute, the change in instantaneous frequency of the fluorescence is proportional to the change in free energy of the solute/solvent system. From this we can construct a new solvent response function

$$C(t) = \frac{\omega(t) - \omega(\infty)}{\omega(0) - \omega(\infty)}$$

Equation 3.10

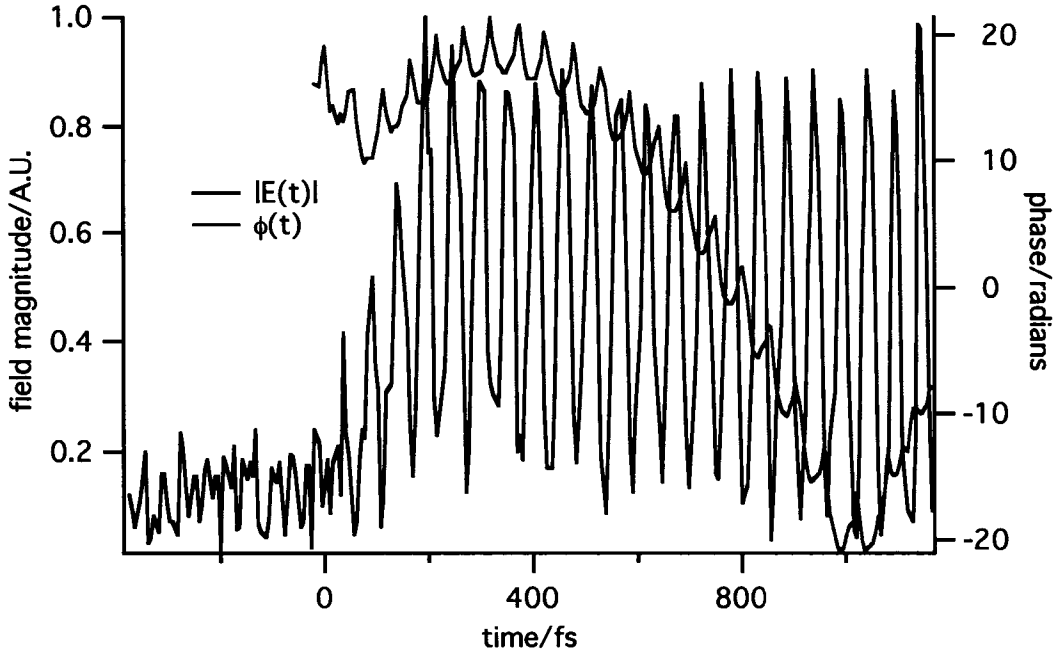


Figure 3.13 Magnitude and phase representation of the $E(t)$ returned from the phase retrieval algorithm of the fluorescence signal from C153 in acetonitrile at crystal angle 59°.

The plots of the magnitude and phase in Figure 3.13 exhibit rapid modulations due to source of the modulation is the high bandwidth of the signal. Because the fluorescence signal is so broad, the many frequencies existing at the same time interfere with each other. The interference can cause nearly complete cancellation of the electric field at regular intervals, as can be seen in the magnitude plot. Therefore, a direct calculation of the instantaneous frequency cannot be made because it requires the calculation of the time derivative of the phase, which has discontinuities that coincide with the troughs of the magnitude.

Replacing $\omega(t)$ in Equation 3.10 with an intensity weighted average defined over a time interval proportional to the inverse of the fluorescence signal bandwidth helps to combat the discontinuities in $d\phi/dt$. Starting from Equation 3.8 the change in instantaneous frequency over a time interval can be defined as

$$\begin{aligned}\Delta\omega(t,\sigma) &= \omega(t+\frac{\sigma}{2})-\omega(t-\frac{\sigma}{2}) \\ &= \left[\omega_0+\frac{d}{dt}\phi(t+\frac{\sigma}{2})\right]-\left[\omega_0+\frac{d}{dt}\phi(t-\frac{\sigma}{2})\right] \\ &= \left[\frac{d}{dt}\phi(t+\frac{\sigma}{2})\right]-\left[\frac{d}{dt}\phi(t-\frac{\sigma}{2})\right]\end{aligned}$$

Equation 3.11

Then integrating $\Delta\omega(t,\sigma)$ weighted by the intensity over that same interval yields a windowed average frequency

$$\overline{\omega}(\tau,\sigma)=\frac{1}{\sigma}\int_{\tau-\frac{\sigma}{2}}^{\tau+\frac{\sigma}{2}}|E(t)|^2\Delta\omega(t,\sigma)dt$$

Equation 3.12

Making the substitution back into Equation 3.10 yields

$$C_p(t;\sigma)=\frac{\overline{\omega}(t,\sigma)-\overline{\omega}(\infty,\sigma)}{\overline{\omega}(0,\sigma)-\overline{\omega}(\infty,\sigma)}$$

Equation 3.13

for the solvent response function. As mentioned above, the time interval over which $\Delta\omega(t,\sigma)$ will be calculated is proportional to the inverse of the bandwidth of the fluorescence signal. This relation comes from the coherence time of the signal or the length of time between the troughs in the magnitude caused by destructive interference. As the bandwidth increases, the coherence time of the signal decreases.

Figure 3.14 shows the results of performing this analysis on the retrieved electric field from C153/acetonitrile at crystal angle 59° where σ = 53 fs was used. Figure 3.15 shows the re-sampled upconverted fluorescence signal from C153 in methanol at crystal angle 58°. The fluorescence signal is wider for C153 in methanol than in acetonitrile. The $C_p(t)$ curve for methanol is shown in Figure 3.16. Figure 3.17 shows the binomially

smoothed mag-squared complex Fourier transform of the retrieved electric fields. The self-interference of the broadband signals causes the large structural features of the spectra calculated this way.

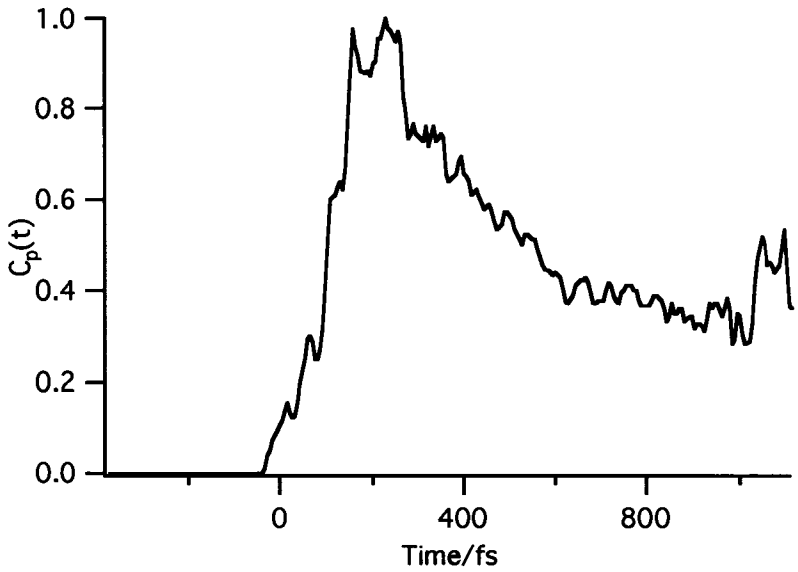


Figure 3.14 $C_p(t)$ for C153/acetonitrile at crystal angle 59°, calculated using σ = 53 fs.

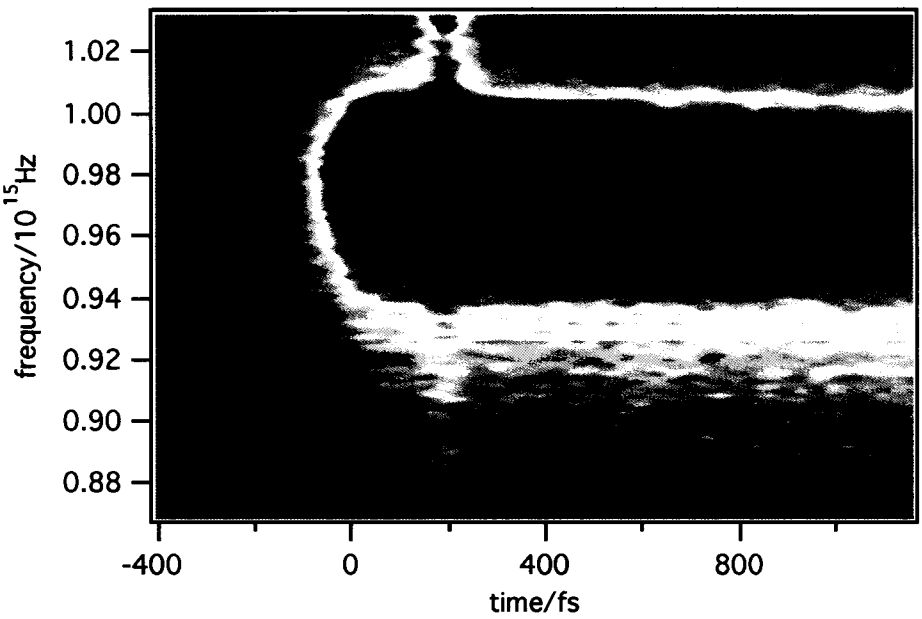


Figure 3.15 Re-sampled cross correlation frequency resolved optically gated fluorescence from C153 in methanol at crystal angle 58°.

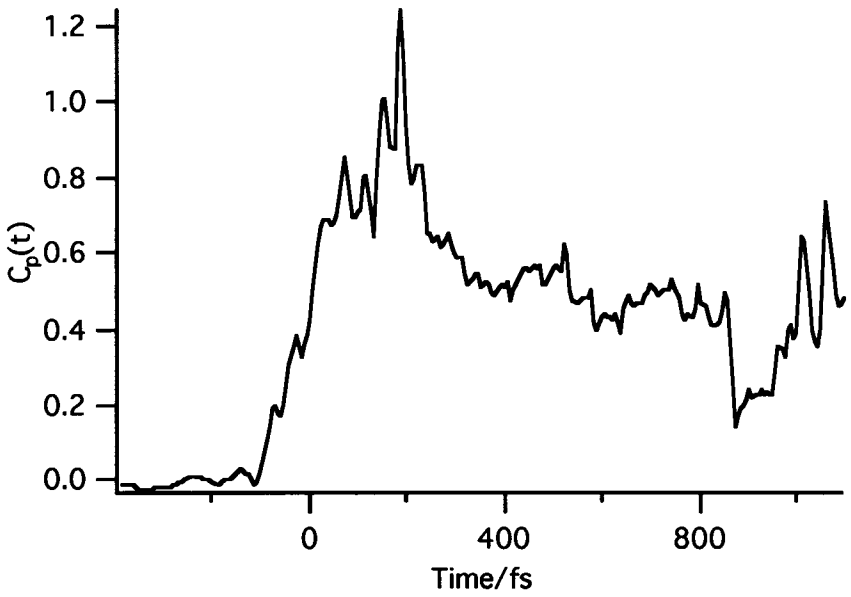


Figure 3.16 $C_p(t)$ for C153/methanol at crystal angle 58°, calculated using σ = 57 fs.

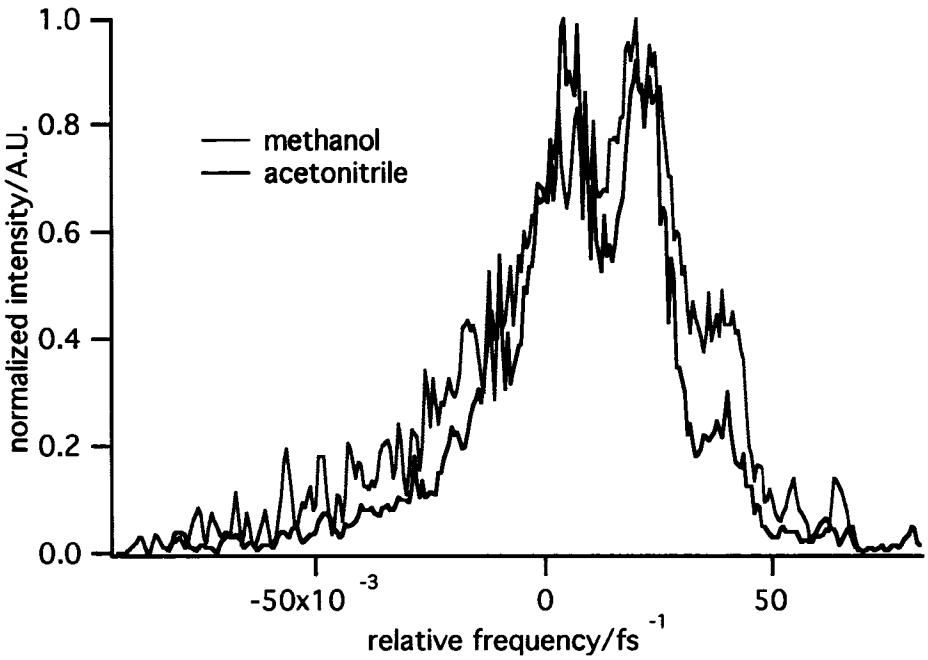


Figure 3.17 Fourier transform of the retrieved fluorescence electric field of C153 in methanol at 58° and acetonitrile at 59

Chapter 4

Results of FROG Simulations

In order to verify the functionality of the phase retrieval algorithm, it was tested on a range of calculated analytical pulses and simulated time resolved fluorescence Stokes shift FROG traces. First, several analytical pulses were used to produce simulated FROG traces to demonstrate that the phase retrieval algorithm produces the correct result. Next, simulated time resolved fluorescence Stokes shift experiments were used to produce simulated FROG traces to show the viability of using phase retrieval methods to measure time resolved fluorescence Stokes shifts free from the influence of the gate pulse.

Analysis of FROG traces produced from coherent pulses, that is, pulses that have a well defined time-bandwidth relationship, produce results with well-known magnitude and phase characteristics. This type of signal is described with a single carrier frequency and a continuously differentiable phase over its entire duration and is present in laser pulses where the stimulated emission synchronizes the phase of the photons that comprise the pulse. This is in contrast to incoherent sources such as spontaneous emission from electronically excited dye molecules in solution, whose emission is not well described by a single carrier frequency and smoothly differentiable phase over its entire duration because there is no external phase synchronization. As a consequence, this type of

emission does not have a definite time-bandwidth relationship. Using analytical pulses with a well defined time-bandwidth relationship allows us to demonstrate that the phase retrieval algorithm developed effectively analyzes pulses.

The creation of the simulated FROG traces of the analytic pulses and the retrieval of the analytic and non-analytic phases requires that a gate pulse be provided to the retrieval algorithm. The retrieved electric field of one of the gate pulses used in the real experiment was used to create the simulated FROG for the analytical pulses. The temporal phase and magnitude of the real gate pulse is plotted below in Figure 4.1.

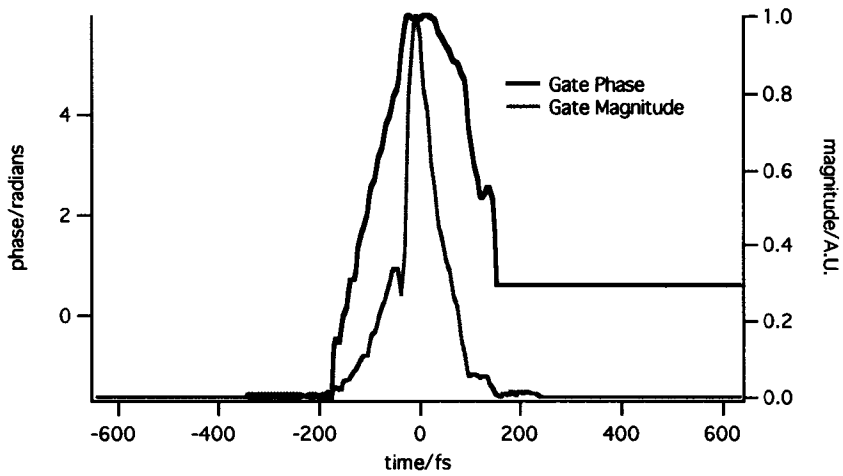


Figure 4.1 The temporal phase and magnitude of the experimentally determined gate pulse electric field. This gate pulse electric field was used to create the simulations of the analytic pulses and was used in the retrieval of their electric fields as well as in the retrieval of the simulated time resolved fluorescence Stokes shift FROG traces.

Simulations of the FROG signals from these analytical pulses were accomplished by the following procedure. First, polar representations (amplitude and phase angle) of the analytic pulses were constructed by defining phase and magnitude functions. Then the polar representation was converted into rectangular ($a+bi$) using Igor Pro's built-in function polar to rectangular converter. Next, the sum frequency signal product matrix

$$\tilde{\mathcal{E}}_s(t,\tau) = \tilde{\mathcal{E}}_g(t) \times \tilde{\mathcal{E}}_f(t-\tau)$$

Equation 4.1

was formed from the gate and the simulated pulse. $\tilde{\mathcal{E}}_g(t)$ is the electric field of the gate pulse and $\tilde{\mathcal{E}}_f(t-\tau)$ is the electric field of the simulated pulse. The analytical pulses were constructed using

$$\tilde{\mathcal{E}}(t) = \mathcal{E}(t)e^{i\phi(t)}$$

Equation 4.2

where $\mathcal{E}(t)$ is the temporal magnitude and $\phi(t)$ is the temporal phase.

Finally, the one-dimensional Fourier transform shown in equation 4.3 was calculated for constant values of τ using Igor Pro's built-in Fast Fourier transform (FFT),

$$\mathcal{F}\{\tilde{\mathcal{E}}_s(t,\tau)\}_\tau = \int_{\infty}^{\infty} \tilde{\mathcal{E}}_s(t,\tau)e^{-i\omega t}dt$$

Equation 4.3

The one-dimensional Fourier transform followed by squaring the resulting magnitude is the mathematical representation of experimentally frequency resolving the sum frequency signal at a constant value of the delay (τ) between the gate pulse and the test pulse (or fluorescence signal).

$$I_{FROG}(\nu,\tau) = \left|\frac{1}{2\pi}\mathcal{F}\{\tilde{\mathcal{E}}_s(t,\tau)\}_\tau\right|^2$$

Equation 4.4

This results in the simulated FROG trace shown in the Figure 4.3 (upper left corner). The simulated FROG trace produced by this procedure is properly sampled in time and frequency for the FFT used in the phase retrieval algorithm. The total time window, $\Delta\tau$ or Δt, is related to the frequency window ($\Delta\nu$) by the following relation

$$\Delta t = \frac{N}{\Delta \nu}$$

Equation 4.5

where N is the number of points in one dimension of the sampled FROG trace.

The simulated FROG trace along with the electric field of the gate pulse is then written to a data file by a custom Igor Pro function in the same fashion as the experimental FROG trace. An initial guess for the phase retrieval algorithm was produced from a Gaussian magnitude and random phase ranging from –1 to 1 and then written to a separate data file. Then the polar representation was converted to the rectangular representation. The real and imaginary parts of the rectangular form of the initial electric field guess are plotted in Figure 4.2. This initial guess was used for all retrievals of simulated pulses and FROG traces to eliminate any dependence on the initial guess.

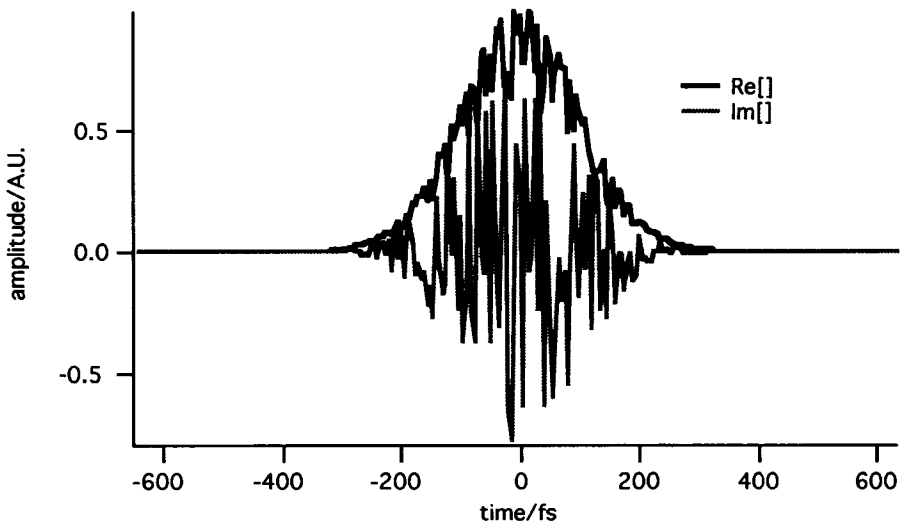


Figure 4.2 The real and imaginary parts of the initial electric field guess supplied to the phase retrieval algorithm.

Each simulated FROG trace was calculated on a 256 x 256 point grid. The retrieval was set to run for 10,000 iterations. An error metric was calculated for each iteration using[29]

$$Error = \sqrt{\sum_{\nu,t} [I_{FROG}(\nu,t)_{in} - \mu \times I_{FROG}(\nu,t)_{calc}]}$$

Equation 4.6

where $I_{FROG}(\nu,t)_{in}$ is the simulated FROG trace, $I_{FROG}(\nu,t)_{calc}$ is the trial FROG trace calculated by the algorithm, μ is a constant scalar that is adjusted by the algorithm to minimize the error for each iteration and the sum runs over all grid points. The electric field of the iteration with the lowest error was saved along with the field of the last iteration. The electric field of the lowest iteration with the lowest error is reported here as the result or the retrieved electric field.

Simulation of Analytic Pulses

Gaussian Magnitude with Zero Phase

The first test of the phase retrieval algorithm was to retrieve the phase of a pulse with zero or flat phase. This pulse was constructed from a Gaussian magnitude with 80 fs width and zero phase.

$$\mathcal{E}(t) = \frac{1}{80\sqrt{2\pi}} Exp\left\{-\frac{1}{2}\left(\frac{t}{80}\right)^2\right\}$$

Equation 4.7

$$\phi(t) = 0$$

Equation 4.8

The simulated FROG trace and the results from the phase retrieval algorithm are shown in Figure 4.3. The upper left panel shows the simulated FROG trace calculated from the

zero phase test pulse and the gate pulse shown in Figure 4.2. The upper right panel shows the retrieved FROG trace. The bottom left panel shows the phase of the input (test) and output (retrieved) pulse. The bottom right shows the magnitude of the input and output pulses.

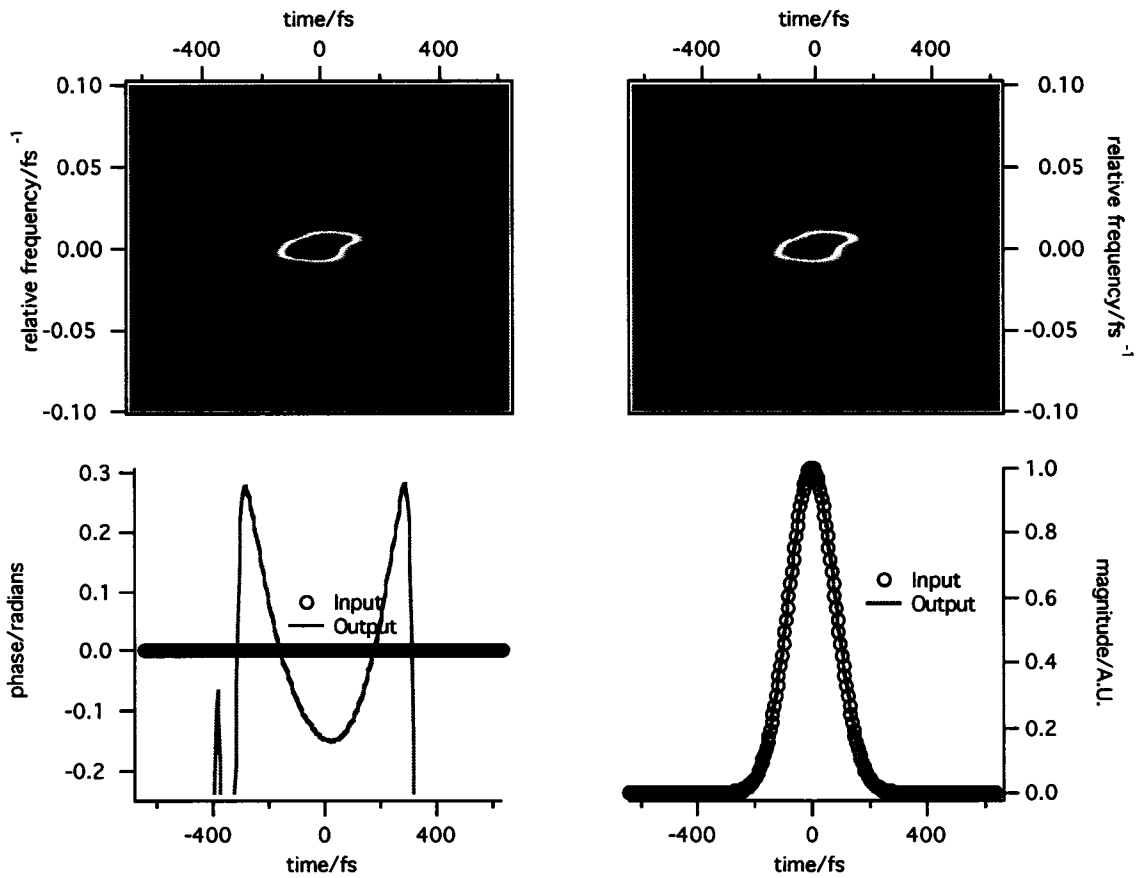


Figure 4.3 From upper left continuing clockwise: Simulated FROG trace of a pulse with Gaussian magnitude and no (or flat) phase, the retrieved FROG trace, the temporal magnitude of the input pulse (Input) and the retrieved magnitude (Output), the temporal phase of the input pulse (Input) and the retrieved phase (output).

The resulting retrieved phase, which has been shifted vertically by a constant (lower left panel Fig. 4.3) exhibits some curvature that was not present in the input pulse. However, the total excursion of the curvature over the portion of the pulse with any

appreciable magnitude is less than 0.45 rad. As will be seen in subsequent simulations, this would not be detectable in other cases. The amplitude of the curvature can be used as an estimate of the error or the uncertainty in the retrieved phase. The retrieved magnitude shows no sign of any deviation from that of the input pulse. Additionally, the retrieved FROG trace (upper right panel) is nearly indistinguishable from the simulated trace.

Gaussian magnitude with quadratic phase (negative sign)

Our second test of the phase retrieval algorithm was performed on a pulse with negative quadratic temporal phase. This pulse was constructed in the same manner as the first pulse, with the temporal magnitude and phase given by Equation 4.9 and Equation 4.10 respectively.

$$\mathcal{E}(t) = \frac{1}{80\sqrt{2\pi}} Exp\left\{-\frac{1}{2}\left(\frac{t}{80}\right)^2\right\}$$

Equation 4.9

$$\phi(t) = -\frac{t^2}{1000}$$

Equation 4.10

The simulated FROG trace was produced from this pulse along with the gate pulse mentioned above and shown in Figure 4.4. Because the instantaneous frequency depends on the first time-derivative of the temporal phase $\phi(t)$, the frequency should decrease linearly with time. This decrease in frequency with respect to time can be seen in the simulated FROG trace as a skewing of the signal from upper left (higher frequency) to the lower right (lower frequency) of the trace.

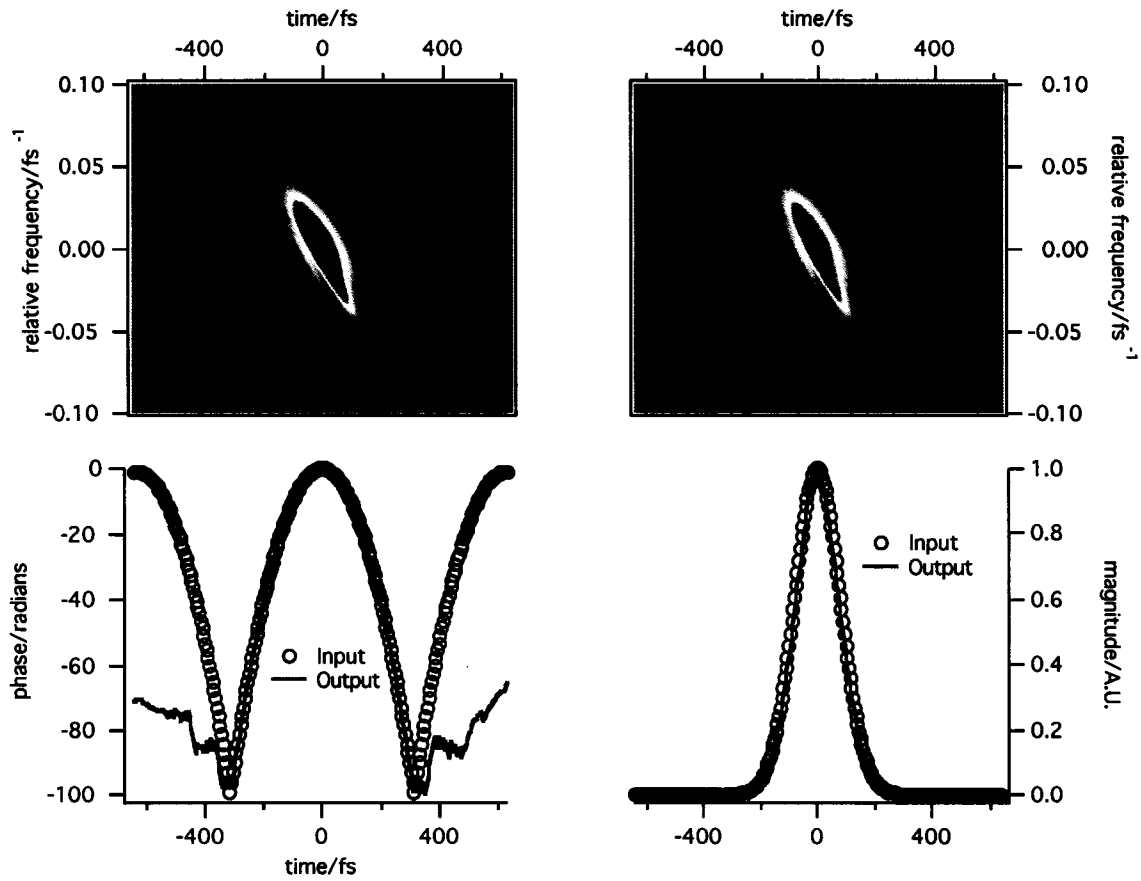


Figure 4.4 From upper left continuing clockwise: Simulated FROG trace of a pulse with Gaussian magnitude and negative quadratic phase (according to **Equation 4.9** and **Equation 4.10**), the retrieved FROG trace, the temporal magnitude of the input pulse (Input) and the retrieved magnitude (Output), the temporal phase of the input pulse (Input) and the retrieved phase (output).

Gaussian magnitude with quadratic phase (positive sign)

A pulse with the same quadratically dependent phase but opposite sign was used to create a simulated FROG trace, Figure 4.5. The temporal magnitude of the electric field was the same as that in used in the previous pulse given by Equation 4.9. However the sign of the phase function was positive

$$\phi(t) = \frac{t^2}{1000}$$

Equation 4.11

In this case the signal in the FROG trace should be skewed from lower left to upper right indicating an increase in instantaneous frequency. The increase is due to the positive slope of $d\phi(t)/dt$.

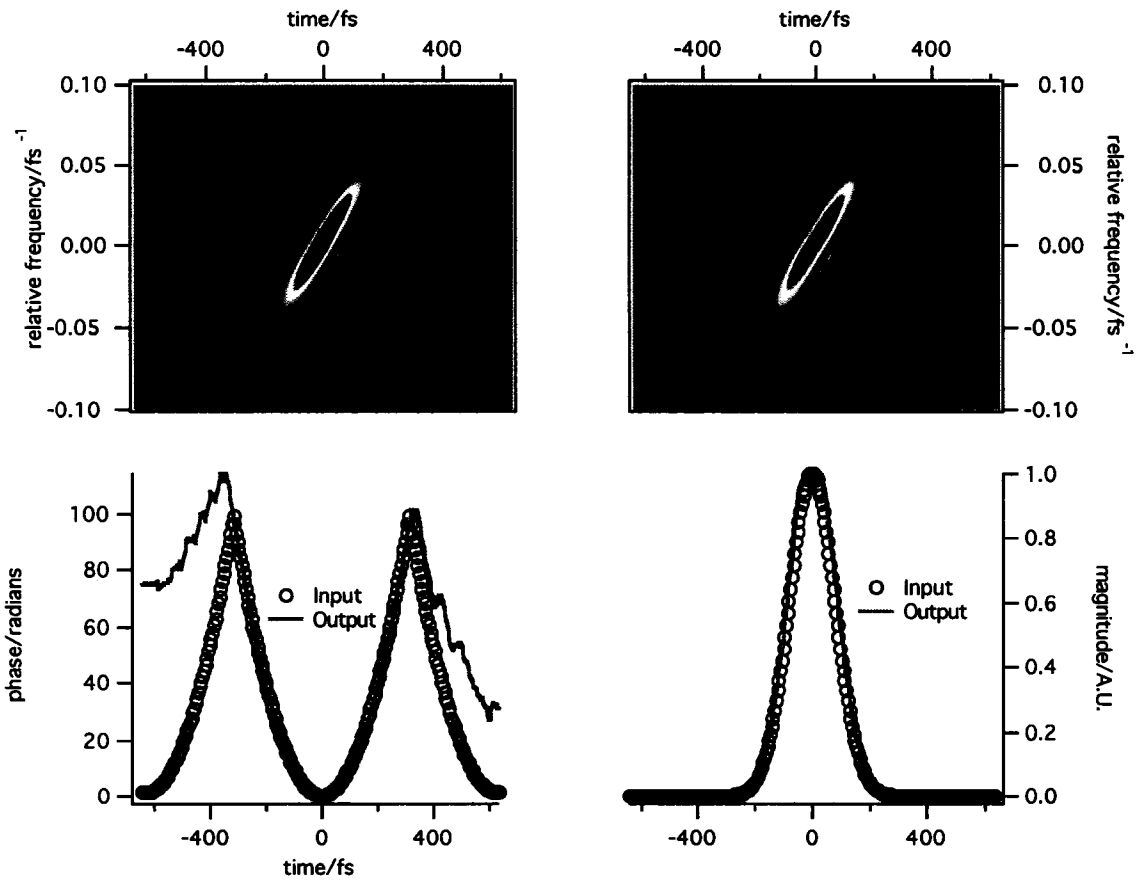


Figure 4.5 From upper left continuing clockwise: Simulated FROG trace of a pulse with Gaussian magnitude and positive quadratic phase (according to **Equation 4.9** and **Equation 4.11**), the retrieved FROG trace, the temporal magnitude of the input pulse (Input) and the retrieved magnitude (Output), the temporal phase of the input pulse (Input) and the retrieved phase (output).

Again, the retrieved FROG trace in Figure 4.5 (upper right panel) is nearly indistinguishable from the simulated trace. The retrieved phase (labeled Output in the

lower right panel of Figure 4.5) lays almost exactly over the phase of the simulated pulse. The retrieved magnitude does, however, show some deviation from that of the simulated pulse at the maximum. It appears to have a broader, slightly non-Gaussian shape near the top. This is most likely due to the retrieval algorithm being trapped in a local minimum in the parameter space. A remedy for this problem is to start the phase retrieval algorithm again with a different starting guess. Since this deviation was somewhat minor and the retrieved phase agreed highly with the phase of the simulated pulse, this was not done.

Gaussian magnitude with cubic phase

The last of the analytic pulse forms to be simulated was a pulse with a Gaussian magnitude and cubic phase. The magnitude for this pulse was twice as wide as the previous pulses and is given in Equation 4.12. The phase is given in Equation 4.13.

$$\mathcal{E}(t) = \frac{1}{160\sqrt{2\pi}} Exp\left\{-\frac{1}{2}\left(\frac{t}{160}\right)^2\right\}$$

Equation 4.12

$$\phi(t) = -\frac{(t-100)^3}{800000}$$

Equation 4.13

The cubic phase results in a pulse with quadratic instantaneous frequency. This quadratic shape can be seen in the simulated FROG trace of this pulse, Figure 4.6 upper left pane. The frequency increases until 100 fs then decreases according to the parabola described by the time derivative of Equation 4.13. The vertex of the parabola appears at t=100 fs.

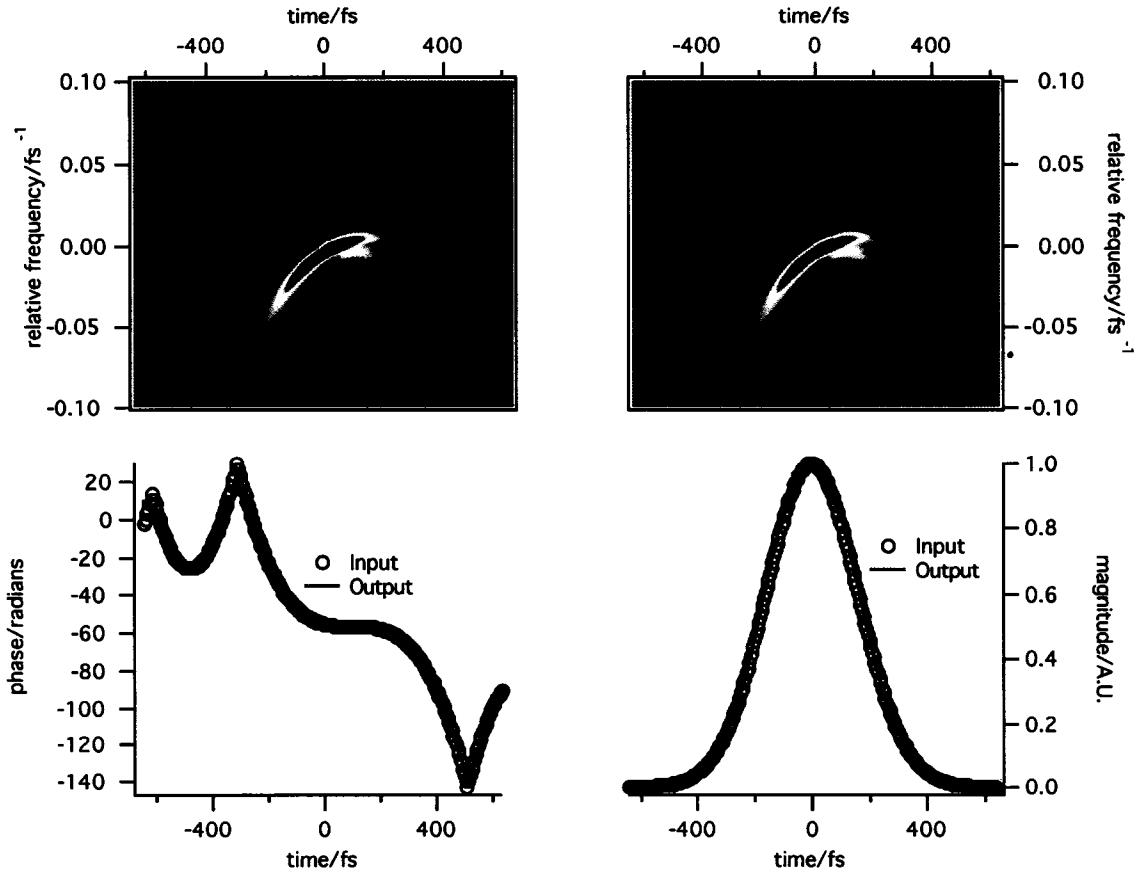


Figure 4.6 From upper left continuing clockwise: Simulated FROG trace of a pulse with Gaussian magnitude and cubic phase (according to eq.), the retrieved FROG trace, the temporal magnitude of the input pulse (Input) and the retrieved magnitude (Output), the temporal phase of the input pulse (Input) and the retrieved phase (output).

Even for this more complex pulse shape, the retrieved FROG trace (upper right panel of Figure 4.6) looks almost identical to the simulated trace. As with the other simulated analytic pulses, the retrieved phase closely matches that of the simulated pulse. The magnitude again seems to be a little wider at the top than the magnitude of the input pulse and the apex is shifted slightly to negative time. Again, these deviations could probably be remedied by using a different starting electric field.

Time Resolved Fluorescence Stokes Shift FROG Simulations

In contrast to the FROG traces formed analytically from temporal phase and magnitude function convolved with a gate pulse, time resolved fluorescence Stokes shift FROG traces were simulated by evaluating an expression for the time resolved fluorescence signal.

$I_{FROG}(\nu, t)$

Equation 4.14

At a constant time t_c, $I_{FROG}(\nu, t_c)$ can be described with an appropriate lineshape function such as a Gaussian, sum of Gaussians, log-normal or sum of log-normals whose parameters can be time dependent. The Gaussian and sum of Gaussian lineshape functions were chosen to simulate these time resolved fluorescence signals.

$$I_{FROG}(\nu; t_c) = A_{t_c} Exp\left[-\left(\frac{\nu - \nu_{0;t_c}}{width}\right)^2\right]$$

Equation 4.15

For instance, Equation 4.15 can be evaluated for various delay times and a simulated FROG trace can be produced. Producing FROG traces by directly creating time and frequency resolved intensities allows us to explore the behavior of the phase retrieval algorithm on traces that violate the time-bandwidth product relationship.

Simulated Time Resolved Fluorescence FROG Trace with no Stokes Shift

An example of a FROG trace produced this way can be seen in the upper left panel of Figure 4.7. Here, there was no change in the Gaussian peak frequency so the

signal shows no Stokes shift. The amplitude of the Gaussian lineshape function (A_{t_c}) was determined by the function plotted in lower right panel of Figure 4.7 and labeled as input. This amplitude function was made up of a Gaussian rise until –400 fs, a Gaussian fall starting at 400 fs and a constant middle portion.

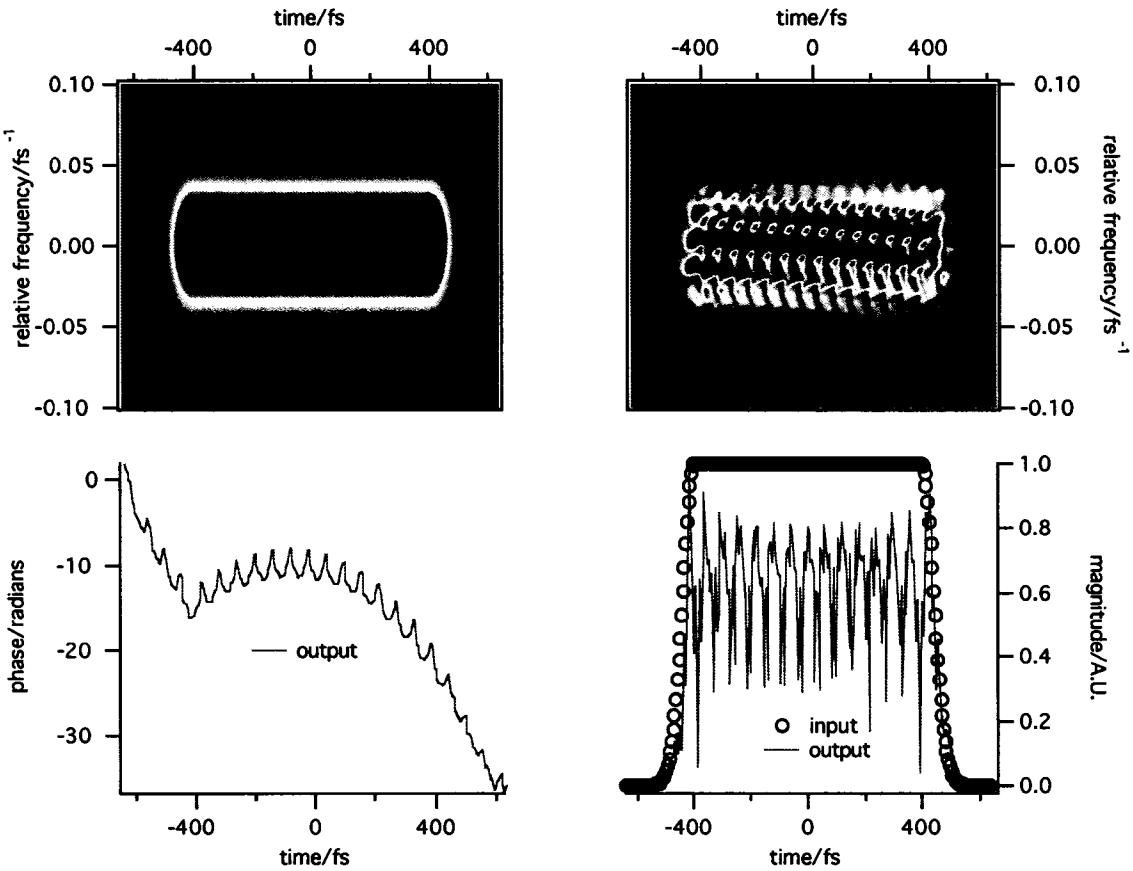


Figure 4.7 Simulated time resolved fluorescence Stokes shift FROG trace and its retrieval. Clockwise starting from upper left: the simulated FROG trace created by using Equation 4.15, the retrieved FROG, the simulated (labeled input) and retrieved (labeled output) magnitude, and retrieved phase.

One striking feature of the retrieved FROG trace and of the retrieved phase and magnitudes is the modulation of each where none exists in the input FROG trace. This stems from the way the simulated FROG trace was produced. In contrast to the analytic

pulses from the previous section, there is no time-bandwidth product to relate the duration of the pulse to the measured bandwidth because the signal was not produced using a function of the form

$$\tilde{\mathcal{E}}_s(t,\tau) \propto \tilde{\mathcal{E}}_g(t) \times \tilde{\mathcal{E}}_f(t-\tau)$$

Equation 4.16

where $\tilde{\mathcal{E}}_g(t)$ and $\tilde{\mathcal{E}}_f(t)$ are of the form

$$\tilde{\mathcal{E}}(t) = \mathcal{E}(t)e^{i\phi(t)}$$

Equation 4.17

However, the phase retrieval algorithm does assume that the signal can adequately be described by Equation 4.16. Normally the bandwidth exhibited by a FROG trace of an analytic pulse with duration on the order of one ps (such as that of the simulated pulse pictured in Figure 4.7) would be 100-200 times narrower than that of the non-analytic pulse pictured in Figure 4.7, or about 4 x 10^{-4} fs^{-1}. In order for the algorithm to attempt to match the larger bandwidth of the simulated FROG signal, it must introduce modulation in the temporal magnitude that is not present in the input FROG trace. The rate of the modulation will be somewhat dependent on the bandwidth of the input FROG trace, i.e. the larger the bandwidth, the more rapid the modulation.

Another feature of the retrieved pulse is the curvature of the phase. Clearly, the slope of the phase should be constant if there is no change in the frequency of the FROG trace. Similar to the flat phase case from the previous section (Figure 4.3) the total excursion of the retrieved phase can serve as an estimate of the upper limit to the uncertainty of determining the phase of these non-analytic signals. The total change in the phase (not considering the rapid scallops), $\Delta\phi(t)$, over the range where there is

appreciable signal magnitude starting at –450 and ending at 450 fs is approximately 15 rad. As with examples from previous sections, it is possible that better retrieval results are possible with the use of a different starting guess.

Simulated Time Resolved Fluorescence FROG Trace with Stokes Shift and Gaussian Rise and Fall

To simulate a dynamic fluorescence signal a Stokes shift function was created to determine the peak frequency of the Gaussian lineshape, Equation 4.18. In Equation 4.18, the total shift is determined by the pre-exponential factor of 0.06 fs^{-1}, which is approximately 2000 cm^{-1}.

$$\nu_0(t) = 0.06 \times Exp\left(-\frac{t+400}{200}\right) - 0.03$$

Equation 4.18

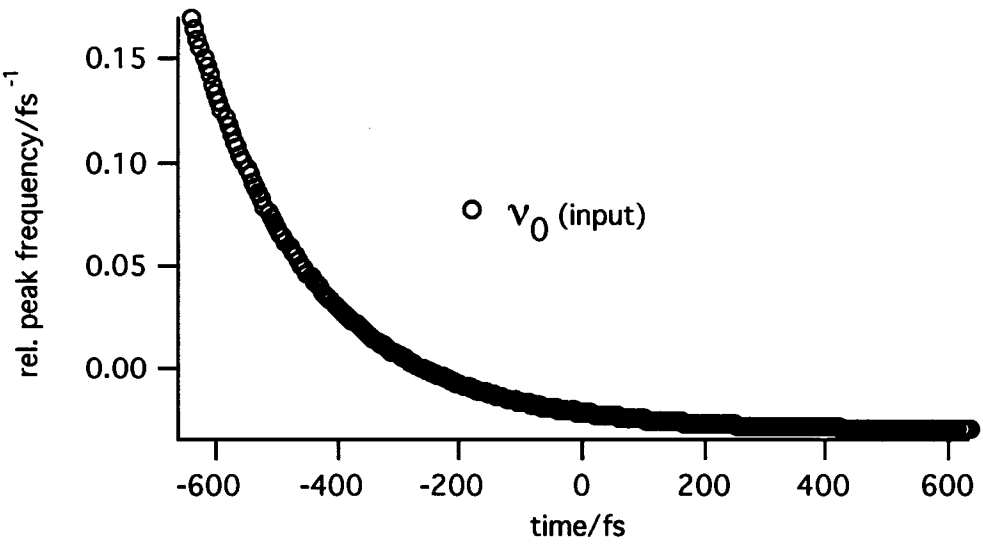


Figure 4.8 Function used to describe the center frequency (ν_0) of the Gaussian lineshape function as a function of time, or time resolved Stokes shift. This function was used to produce the simulated time resolved fluorescence Stokes shift FROG trace.

In order to relate this dynamic shift of the peak frequency to the phase, the equation for the instantaneous frequency can be integrated to solve for $\phi(t)$. Equation 4.19 shows how the experimentally measured or simulated frequency expresses as the Stokes shift with units of cycles/sec can be related to the retrieved phase. Starting from the expression for the instantaneous frequency, $\omega(t)$, in rad/sec, we can integrate to solve for the temporal phase.

Next, we integrate the Stokes shift, $\nu(t)$, and convert the result from cycles/sec to rad/sec by multiplying by 2π.

$$\begin{aligned} \omega(t) &= \omega_0 + \frac{d\phi(t)}{dt} \\ \int \omega(t)dt &= \int \omega_0 + \frac{d\phi(t)}{dt}dt \\ &= \phi(t) + const. \end{aligned}$$

$$\begin{aligned} 2\pi\nu(t) &= \omega(t) \\ 2\pi \int \nu(t)dt &= \int \omega(t)dt \\ 2\pi \int \nu(t)dt &= \phi(t) + const. \end{aligned}$$

Equation 4.19

The upper left panel of Figure 4.9 shows the simulated FROG trace of a signal whose temporal magnitude is the same as the signal shown in Figure 4.7, but instead of having constant frequency as in the previous example, the peak frequency is determined by the Stokes shift described by Equation 4.18 and plotted in Figure 4.8. The Stokes shift was integrated and multiplied by 2π and is plotted with the retrieved phase in the lower left panel of Figure 4.9.

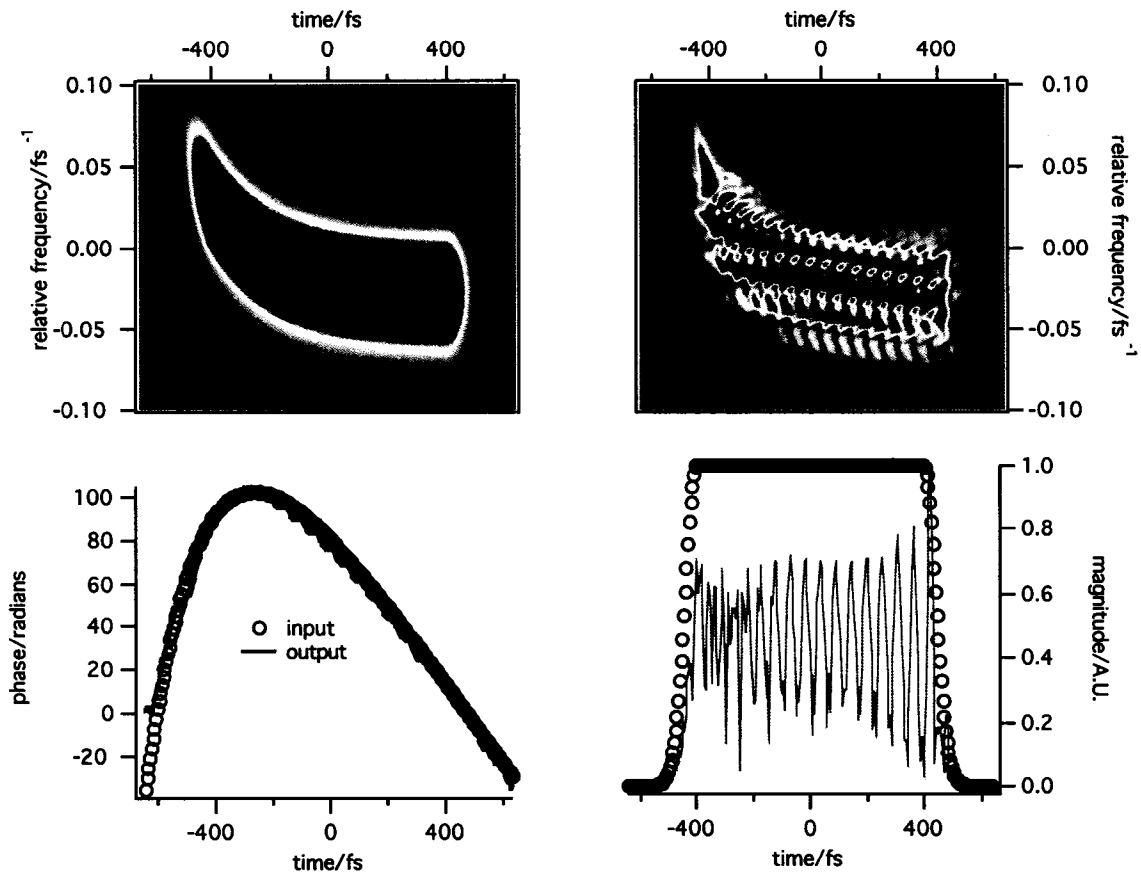


Figure 4.9 Simulated time resolved fluorescence Stokes shift FROG trace of a dynamical spectrum with Stokes shift shown in Figure 4.8. Clockwise starting from upper left: the simulated FROG trace created by using Equation 4.15, the retrieved FROG, the simulated (input) and retrieved (output) magnitude. The lower left panel shows the time integrated Stokes shift from Figure 4.8 (labeled input) and the retrieved phase (labeled output).

The retrieved FROG trace shows the same kind of modulation as the previous example without a Stokes shift. This modulation is also evident in both the retrieved magnitude and phase (lower panels of Figure 4.9). However, since the simulated FROG trace was constructed with a Stokes shift, the retrieved phase is not expected to be flat or have a constant slope according to Equation 4.19. The lower left panel shows the retrieved phase labeled output and the integrated Stokes shift labeled input. There is substantial agreement between the two curves.

Exact agreement between the retrieved phase and the integrated Stokes shift curves will never be achieved. The reason for this stems from the method by which the simulated FROG trace is constructed. Directly constructing a FROG trace by calculating $I_{FROG}(\nu, t)$ for every point on the simulation grid with the use of a lineshape function and a Stokes shift bypasses the mathematical representation of the sum frequency gating that provides the time resolution of this measurement. Because the phase retrieval algorithm assumes that the sum frequency FROG trace is influenced by both the fluorescence and gate pulses, the retrieved phase will reflect the attempt to deconvolve the influence of the gate even though none was explicitly used in the construction of the original FROG trace.

Simulated Time Resolved Fluorescence FROG Trace with Stokes Shift and Gaussian Intensity Rise without Fall

The simulation below (Figure 4.10) uses the same Stokes shift and Gaussian intensity rise as the previous simulation. However, the signal intensity was not rolled off with a Gaussian fall, but instead was allowed to continue until the edge of the trace. Eliminating the fall or roll off of the signal intensity was done to investigate the effect of a sudden truncation of the FROG signal on the retrieved phase and magnitude.

Again, the retrieved FROG trace, temporal magnitude and phase show modulation similar to that in the retrievals of the other traces produced by direct calculation of $I_{FROG}(\nu, t)$. While the retrieve phase and magnitude are somewhat different from that of the results from the simulation with rolled off intensity, the differences are not drastic. Most importantly, the retrieved phase is only slightly different after about 250 fs from that of the previous simulation. While this could be an artifact of the truncation, it could

also be due the increased signal duration and its effect on the singular value decomposition.

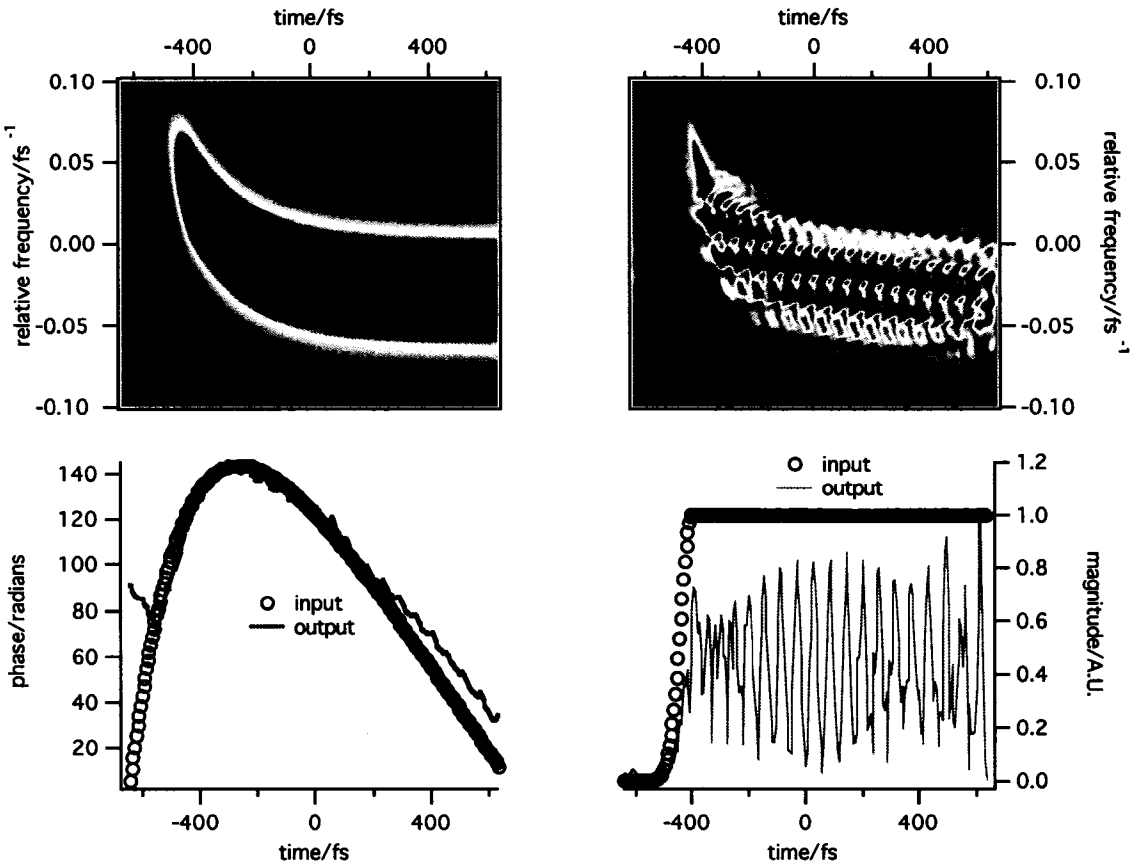


Figure 4.10 Simulated time resolved fluorescence Stokes shift FROG trace of a dynamical spectrum with Stokes shift shown in Figure 4.8. Clockwise starting from upper left: the simulated FROG trace created by using Equation 4.15, the retrieved FROG, the simulated (input) and retrieved (output) magnitude. The lower left panel shows the time integrated Stokes shift from Figure 4.8 (labeled input) and the retrieved phase (labeled output).

One possible source for the truncation artifacts could result from the Fourier transform and inverse Fourier transform steps. Recall, that within the phase retrieval algorithm the FROG signal is calculated by a one-dimensional Fourier transform of the two-dimensional result of convolving the gate and fluorescence pulse electric fields. This results in a matrix sampled in frequency and time. The frequency axis is the spectrum of the sum frequency signal for a constant value of the delay between the gate and the start

of the fluorescence pulse. The time axis is the temporal profile of the fluorescence pulse. The next step in the phase retrieval algorithm is to substitute the magnitude (as the square root of the intensity) of the input trace for the magnitude of the calculated signal leaving the phase untouched, which is where the intensity of the input trace (truncated or otherwise) enters the algorithm. The next step is to inverse Fourier transform this matrix along the frequency axis. Because the Fourier and inverse Fourier transforms are performed along the axis derived from the temporal profile of the gate pulse, and the total temporal window for this axis is always several times larger than the temporal width of the gate pulse, there is never a Fourier or inverse Fourier transform of a time truncated signal. Thus, artifacts from signal intensities truncated along the time dimension will not effect the retrieved phase.

Simulated Time Resolved Fluorescence FROG Trace with Stokes Shift and Gaussian Intensity Rise with Exponential Decay

In these simulations the Stokes shift function and intensity rise used were the same as in the previous two simulations. However, instead of a region of constant intensity as in the previous examples, the intensity decreased exponentially.

In the first simulation using an exponentially decaying intensity, the lifetime, or 1/e time, of the signal was 200 fs and began at the apex of the Gaussian rise at –400 fs, see Equation 4.20. The simulated FROG trace can be seen in the upper left panel of Figure 4.11.

$$I(t) = \begin{cases} exp\left[-\left(\frac{t+400}{40}\right)^2\right] & t \leq -400\text{fs} \\ exp\left[-\frac{t+400}{200}\right] & t > -400\text{fs} \end{cases}$$

Equation 4.20

As with the previous simulations, the general shape of the integrated Stokes shift is reproduced fairly well over the entire time window of the FROG trace. The highest agreement is in the region of where there is appreciable intensity, from –450 to approximately 200 fs. There seems to be lower agreement in the input and output intensity profiles, however. Following the arguments from the previous section regarding the lack of influence on the retrieved phase of FROG signal truncation, it is possible that a better starting guess for the retrieval algorithm might produce a better retrieved magnitude and/or phase.

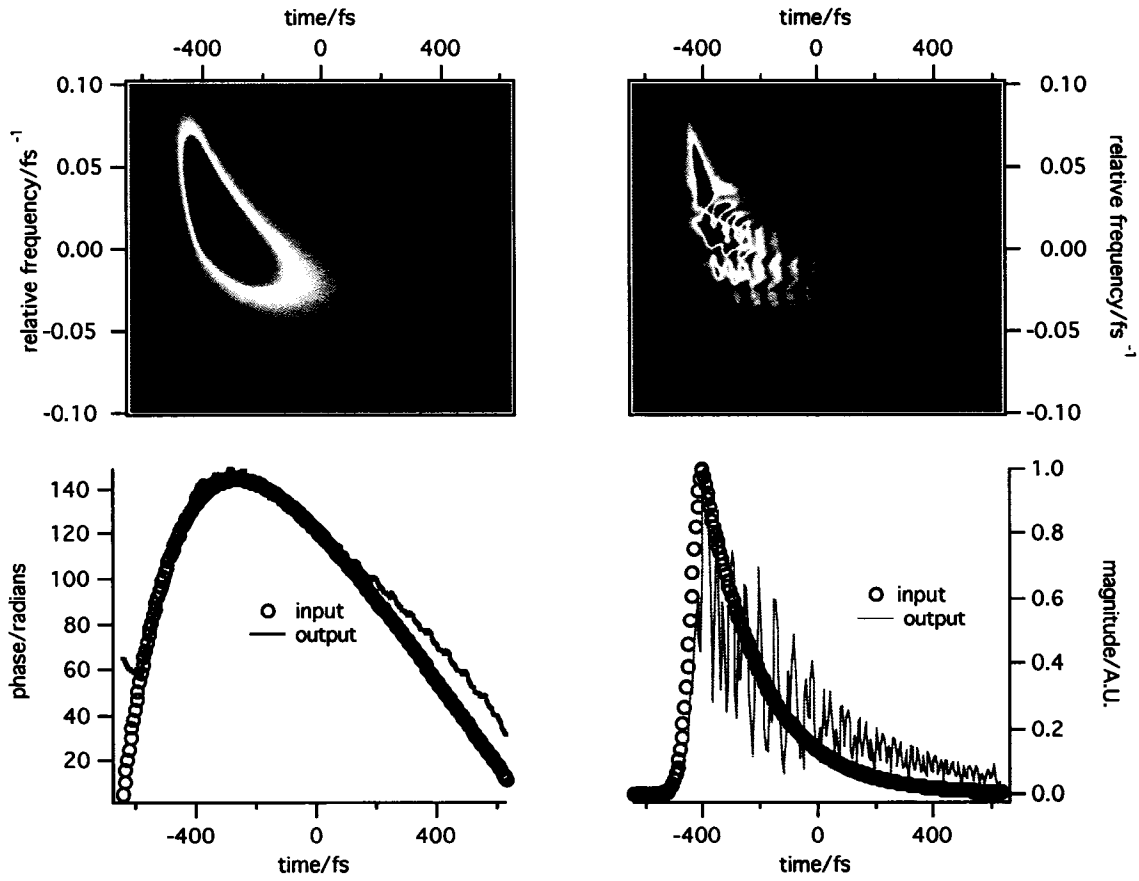


Figure 4.11 Simulated time resolved fluorescence Stokes shift FROG trace of a dynamical spectrum with Stokes shift shown in Figure 4.8 and an exponentially decaying intensity. Clockwise starting from upper left: the simulated FROG trace created by using Equation 4.15 and Equation 4.20, the retrieved FROG, the simulated (input) and retrieved (output) magnitude. The lower left panel shows the time integrated Stokes shift from Figure 4.8 (labeled input) and the retrieved phase (labeled output).

The second simulation of a signal with an exponentially decaying intensity used a decay with a longer lifetime of 1000 fs, see Equation 4.21. The FROG trace simulated in this way is plotted in the upper left panel of Figure 4.12.

$$I(t) = \begin{cases} exp\left[-\left(\frac{t+400}{40}\right)^2\right] & t \leq -400\text{fs} \\ exp\left[-\frac{t+400}{1000}\right] & t > -400\text{fs} \end{cases}$$

Equation 4.21

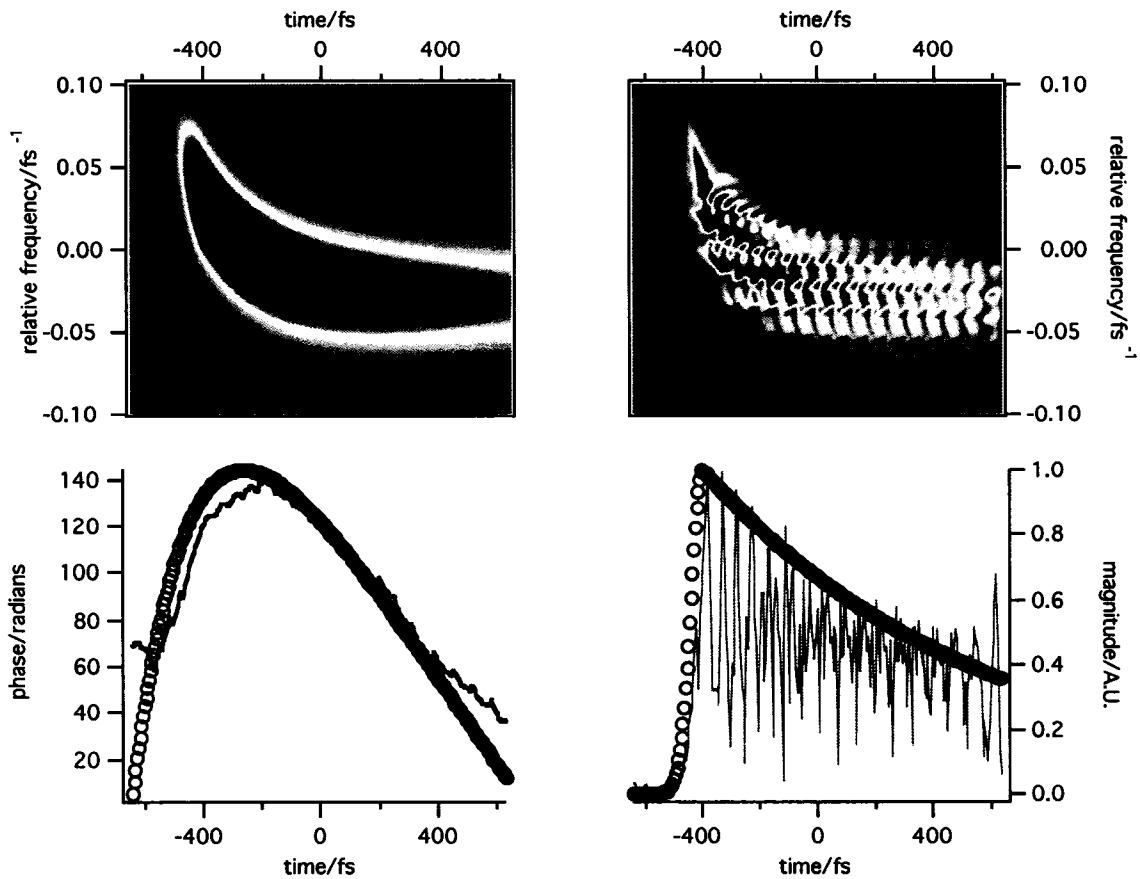


Figure 4.12 Simulated time resolved fluorescence Stokes shift FROG trace of a dynamical spectrum with Stokes shift shown in Figure 4.8 and an exponentially decaying intensity. Clockwise starting from upper left: the simulated FROG trace created by using Equation 4.15 and Equation 4.21, the retrieved FROG, the simulated (input) and retrieved (output) magnitude. The lower left panel shows the time integrated Stokes shift from Figure 4.8 (labeled input) and the retrieved phase (labeled output).

The retrieved phase and magnitude have the correct overall shape, however, they do not match the integrated Stokes shift and input magnitude as closely as the other simulations. Again, this seems to point to the need for a better initial guess for the phase retrieval algorithm.

Simulated Time Resolved Fluorescence FROG Trace with Stokes Shift Derived from Independently Determined Experimental Results

The next simulated FROG trace was constructed to more closely resemble the traces recorded on the cross correlation FROG instrument described in previous chapters. This was accomplished by decreasing the total Stokes shift from approximately 2000 cm^{-1} to approximately 670 cm^{-1}. This reflects the restricted dynamic frequency range of the real cross correlation FROG instrument imposed by the phase matching conditions of the sum frequency crystal. The total Stokes shift of 670 cm^{-1}, or 0.02 fs^{-1}, was chosen because it was typical of a total Stokes shift measured for Coumarin 153 in acetonitrile at a crystal angle of 59° by directly tracking the peak frequency change as a function of time delay between the gate and the fluorescence pulse, as described in previous chapters. A bi-exponential Stokes shift with experimentally determined lifetimes was also employed. The Stokes shift produced this way is given in Equation 4.22 and plotted in Figure 4.13.

$$\begin{aligned} \nu_0(t) &= \left(0.02 \times 0.42 \times Exp\left[-\frac{t+400}{110}\right]\right) \\ &+ \left(0.02 \times 0.58 \times Exp\left[-\frac{t+400}{736}\right]\right) - 0.01 \end{aligned}$$

Equation 4.22

In Equation 4.22 the total frequency shift in fs^{-1} is given by the leading factor of 0.02. The second factor of 0.42 in the first term and 0.58 in the second term are the relative weights of the respective exponential terms. The first exponential term has a lifetime of 110 fs while the second exponential has a lifetime of 736 fs. The spectral width of the simulated signal was also chosen to approximate the spectral width of the real experimentally measured FROG signal. The resulting simulated FROG trace is pictured in the upper left panel of Figure 4.14.

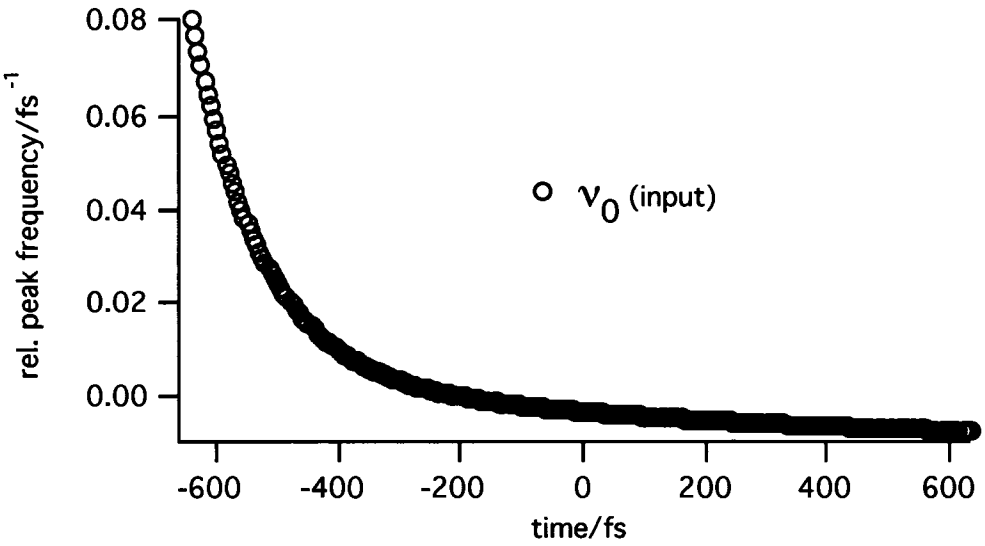


Figure 4.13 Stokes shift used to create the restricted bandwidth time resolved dynamic fluorescence signal FROG trace.

This simulation proved to be the most challenging for the phase retrieval algorithm. There is only a small portion of the retrieved phase that agrees with the integrated Stokes shift curve in the lower left panel of Figure 4.14. The best agreement is in the portion of the signal where the frequency change of the simulated FROG trace is the largest from approximately –450 to –250 fs comprised of about 35 data points. The lack of agreement through the rest of the time window can probably be linked to the

small total Stokes shift in conjunction with the relatively large bandwidth of the simulated FROG signal. It is probably no coincidence that the integrated Stokes shift is best reproduced in the region where the frequency of the FROG signal is changing most rapidly and that the relatively flat region is not well reproduced. As with the previous simulation retrievals, a better starting guess might produce marginally better results.

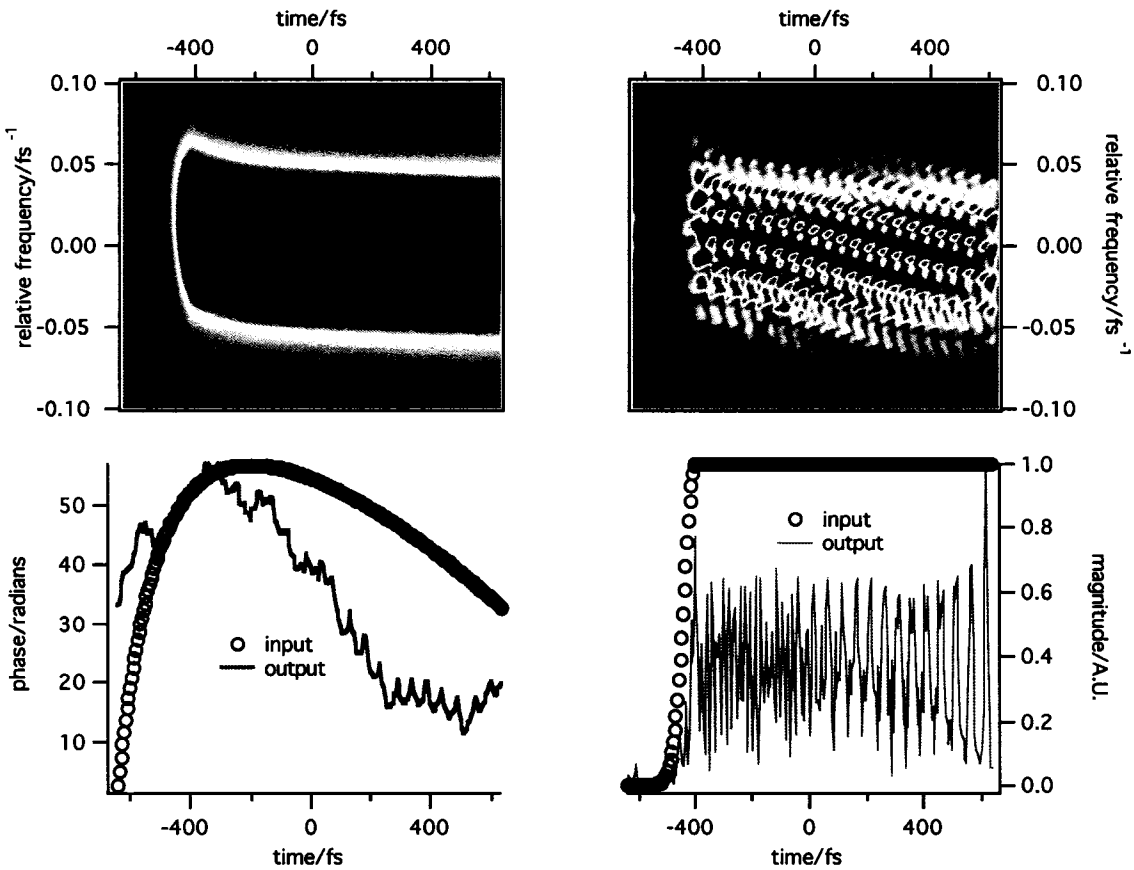


Figure 4.14 Simulation of a FROG trace using Stokes shift parameters determined by an independent analysis method from real experimental data. The Stokes shift was given by Equation 4.22 and plotted in Figure 4.13. Clockwise starting from upper left: the simulated FROG trace created by using Equation 4.15 and Equation 4.22 for the Stokes shift, the retrieved FROG, the simulated (input) and retrieved (output) magnitude. The lower left panel shows the time integrated Stokes shift from Figure 4.8 (labeled input) and the retrieved phase (labeled output).

Calculation of the Intensity Weighted Average Frequency Change for Simulated Time Resolved Fluorescence FROG Traces

In the examples of retrieval of simulated FROG traces described above, the time resolved Stokes shift used to create the FROG trace was known. However, the purpose of performing the FROG experiment is to determine the time resolved Stokes shift. Equation 4.19 shows how the retrieved phase, $\phi(t)$, is related to the Stokes shift, $\nu_0(t)$. It was also shown how integrating the Stokes shift used to create the simulated FROG trace can give an estimation of the retrieved phase. It follows that starting from the retrieved phase, the Stokes shift can be found by calculating the time derivative of the phase according to Equation 4.19.

The retrieved phase from real and simulated time resolved fluorescence FROG traces have discontinuities that produce oscillations in the time derivative. To address the discontinuities, a windowed averaging function was applied to the calculation of the time derivative of the retrieved phase.

$$\overline{\omega}(\tau,\sigma) = \frac{1}{\sigma}\int_{\tau-\frac{\sigma}{2}}^{\tau+\frac{\sigma}{2}} |\tilde{\mathcal{E}}(t)|^2 \Delta\omega(t,\sigma)dt$$

Equation 4.23

This function was derived in chapter 3 and used to calculate the intensity weighted average change in frequency of the dynamic fluorescence signal of Coumarin 153 in acetonitrile and methanol. Here it will be applied in the same way to the retrieved phases for the simulated FROG traces pictured in Figure 4.10 and Figure 4.14. These FROG traces are most like the traces collected for in the real experiment.

The first trace, with a Stokes shift of about 2000 cm^{-1}, is representative of dynamic Stokes shifts of the acetonitrile/Coumarin 153 system measured by other

researchers using the spectral reconstruction method and should largely be free from the bandwidth limitations imposed by the sum frequency gating method utilized in this work.

The average frequency change for this trace and calculated by Equation 4.23 is shown in Figure 4.15. The average frequency was fit to a bi-exponential function between –400 and 400 fs. The fit was started at –400 fs because it is the end of the intensity rise and is usually defined as the de facto start of the Stokes shift when calculated using the spectral reconstruction method. The end of the fit was chosen so as not to include the last 250 fs of data because these were significantly more oscillatory than the rest of the decay and disproportionately skewed the fit. Both single and bi-exponential fits were attempted, but the bi-exponential fit produced a far superior result. The bi-exponential 1/e times with associated fit coefficients are 11 fs (23%) and 206 fs (77%). For this trace, the Stokes shift was defined by Equation 4.18, which was a single exponential function with a 1/e time of 200 fs. It should be noted that while the original Stokes shift was defined with a single exponential function, the Stokes shift serves to define the frequency characteristics of the FROG trace, which is a convolution of the gate and fluorescence pulses and will be influenced by both. So, considering that the simulated FROG trace is influence by both the gate and fluorescence pulses, it is not surprising that the retrieval, is not singly exponential. The influence of the gate pulse will be most evident in the early portion of the trace (in the vicinity of –400 fs), where only part of the gate is temporally overlapped with the fluorescence pulse. Therefore, the addition of a very fast component of 11 fs to the ~200 fs component originating from the Stokes shift is completely reasonable.

This method of comparing the retrieved phase (or, more accurately, the time derivative of the retrieved phase) with the Stokes shift shows that the phase retrieval algorithm is successful at deconvolving the dynamical frequency information of the input signal consisting of a simulated dynamic fluorescence FROG signal with a physically relevant Stokes shift.

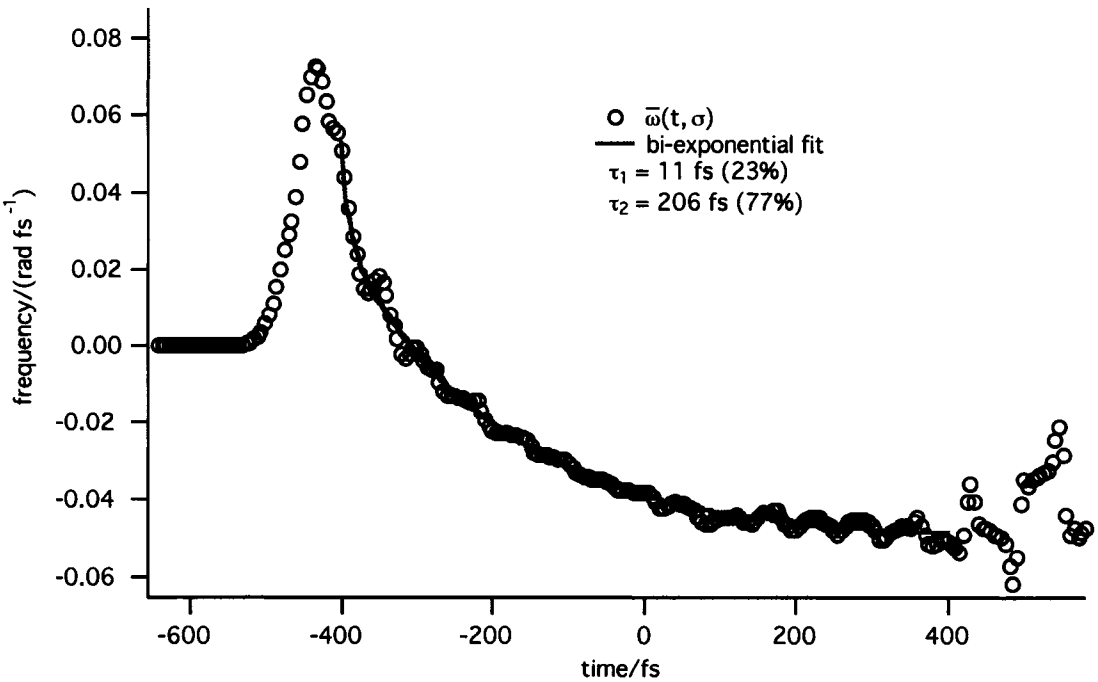


Figure 4.15 Intensity weighted average frequency change calculated from the retrieved phase from the simulated FROG trace pictured in Figure 4.10. The average frequency change was fit with a bi-exponential function (shown in red).

The second trace is representative of the FROG traces collected on the instrument used for this work. The Stokes shift was taken from real acetonitrile/Coumarin 153 experimental data measured using the sum of Gaussians method described in Chapter 3. The Stokes shift used to create the simulated FROG trace is given by Equation 4.22 and

plotted in Figure 4.13. The simulated FROG trace and the retrieved magnitude and phase are plotted in Figure 4.14.

In a similar fashion to the previous trace, with Stokes shift of 2000 cm^{-1}, the intensity weighted average frequency change was calculated from Equation 4.23 with σ = 54 fs. This average frequency change and its fit to an exponential function are plotted in Figure 4.16. A single exponential function was used to fit the average frequency change curve because using a bi-exponential did not provide a better quality fit. Because the curve oscillates significantly though out its entirety, the significance of more exponential terms in the fit would be easily discernable. The fit started at –400 fs for the reasons mentioned above. Since the curve was rather oscillatory through out, no subsequent points where omitted from the curve as they were in the previous example.

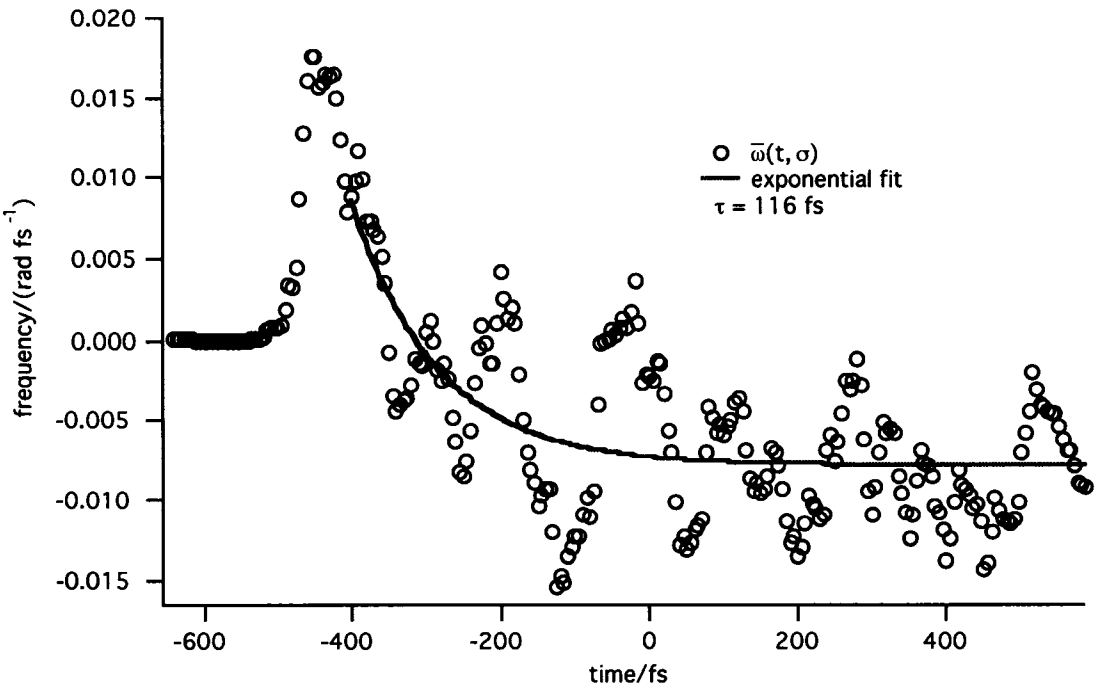


Figure 4.16 Intensity weighted average frequency change calculated from the retrieved phase from the simulated FROG trace pictured in Figure 4.14. The average frequency change was fit with an exponential function (shown in red).

The Stokes shift used to create this simulation was a bi-exponential function with 1/e times of 110 fs (42%) and 736 fs (58%) (see Equation 4.22). The single exponential fit of the calculated average frequency change resulted in a 1/e time of 116 fs. This suggests that for signals with small Stokes shifts like this one, the phase retrieval algorithm can successfully retrieve the dynamic frequency components that result in rapid changes of the peak frequency of the FROG signal, i.e. the faster of the two 1/e times, 110 fs. It is not, however, sensitive to the slower component of 736 fs that results in a much slower change of the FROG signal peak frequency.

Simulations Transitioning from an Analytic Pulse to a Simple Distribution of Emitters

The coherent signals whose FROG traces were simulated in the first section were modeled using Gaussian temporal distributions with second and third order temporal phase functions. This type of signal is commonly used to model the ultra short laser pulses that are produced by Ti:Sapph oscillators. Then the original electric field was successfully retrieved from the simulated FROG trace. In the next section, FROG traces produced by a statistical distribution of emitters were simulated using an empirically derived Gaussian spectral lineshape. The peak frequency of the Gaussian lineshape was varied in time to mimic the dynamic Stokes shift observed in the fluorescence of Coumarin 153 in acetonitrile. The retrieved electric fields and retrieved FROG traces from these signals showed significant modulation that was not present in the original signals. The source of this modulation was proposed to be the violation of the time-bandwidth relation in incoherent signals. To further demonstrate this point a simulated FROG trace that violates the time-bandwidth relation will be constructed from electric

fields that do not violate the time-bandwidth relation. An electric field representing the fluorescence of a single molecule in the gas phase was produced. Then a collection of three of such fields was used to produce a FROG trace of an inhomogeneously broadened fluorescence signal.

In contrast to the previous simulations, the electric fields in this section will be constructed in the frequency domain and then inverse Fourier transformed to produce an electric field in the time domain. To start, the spectral distribution was defined by a Lorentzian lineshape function. The Lorentzian was chosen because it represents the homogeneously broadened spectrum of a single molecule in the gas phase. Equation 4.24 shows the Lorentzian lineshape function $S(\omega)$.

$$S(\omega) = \frac{1}{\pi} \frac{\gamma^2}{(\omega - \omega_0)^2 + \gamma^2}$$

Equation 4.24

where γ is the spectral width, which is related to the excited state lifetime by

$$\gamma \propto \text{lifetime}^{-1}$$

Equation 4.25

and ω_0 is the center frequency. A frequency domain electric field can be constructed using the lineshape function $S(\omega)$ and a form similar to Equation 4.2

$$\tilde{\mathcal{E}}(\omega) = S(\omega) e^{i\varphi(\omega)}$$

Equation 4.26

where $\varphi(\omega)$ is the spectral phase function.

Analytic Pulse from a Lorentzian Lineshape Function

To create the analytic Lorentzian lineshape pulse, Equation 4.24 was used to define a spectrum with $\gamma = 0.011$ fs^{-1} and $\omega_0 = 0$ fs^{-1}. In order to mimic the Stokes shift of the fluorescence of Coumarin 153 dissolved in acetonitrile or methanol, a phase function with a fifth order dependence on frequency was chosen. Any odd ordered phase would have sufficed to give an apparent Stokes shift, but a fifth order phase was found, through trial and error, to give a result that most closely resembled the shifts observed in the real experimental data.

$$\varphi(\omega) = 4000000(\omega - 0.1)^5$$

Equation 4.27

The coefficient of 4000000 was chosen to give a total Stokes shift that was comparable to the shifts of the experimental data. The linear offset of 0.1 fs^{-1} was chosen to give a *temporal* electric field that peaked near –400 fs on the temporal grid, after inverse Fourier transforming the spectral field. Figure 4.17 shows both the spectrum and spectral phase of the spectral electric field defined by Equation 4.24 and Equation 4.27 with $\gamma = 0.011$ fs^{-1} and $\omega_0 = 0$ fs^{-1}. The temporal electric field was produced from this spectral field by inverse Fourier transform.

$$\tilde{\mathcal{E}}(t) = \mathcal{F}^{-1}\{\tilde{\mathcal{E}}(\omega)\}$$

Equation 4.28

Then the simulated FROG trace was formed in the same manner as in previous sections with a Gaussian gate pulse described by Equation 4.29.

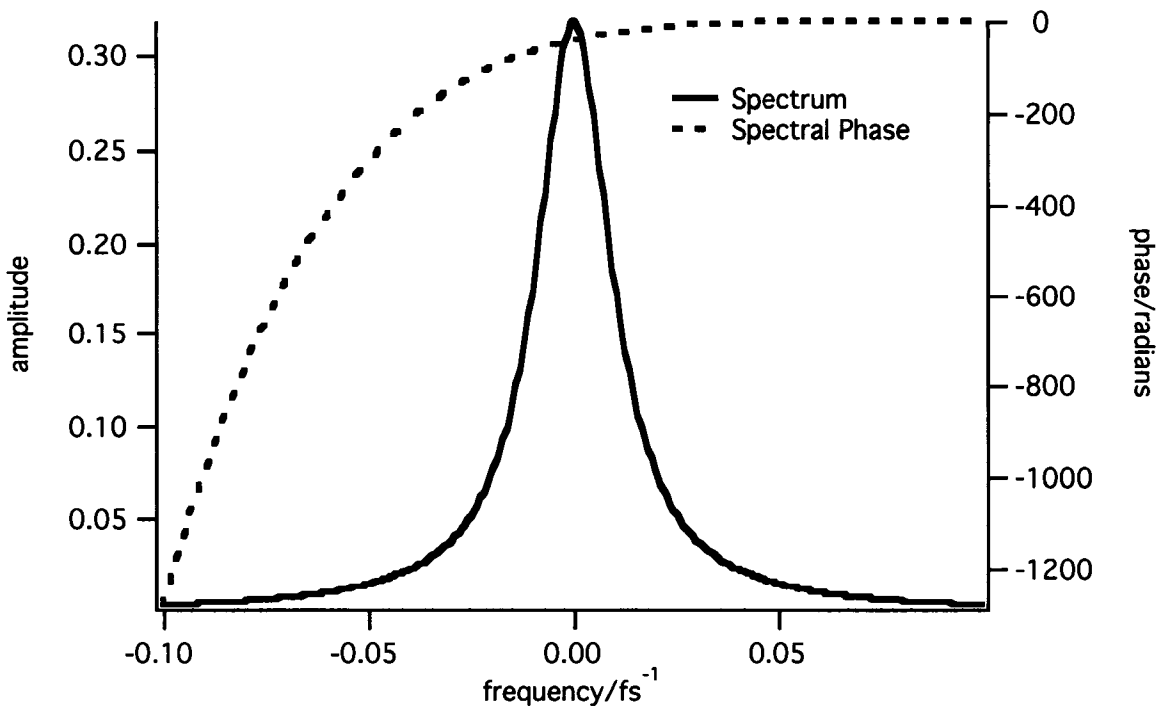


Figure 4.17 The spectrum and spectral phase of the spectral electric field modeling the fluorescence of a single dye molecule in the gas phase. The spectrum and spectral phase are defined by Equation 4.24 and Equation 4.27 with $\gamma = 0.011$ fs^{-1} and $\omega_0 = 0$ fs^{-1}.

In contrast to the simulated FROG traces from the previous sections, the simulated FROG traces using the Lorentzian lineshape function in this section use a Gaussian gate pulse with zero phase as pictured in Figure 4.18. The magnitude and phase are given by Equation 4.29.

$$\mathcal{E}(t) = Exp\left[-2\left(\frac{t}{60}\right)^2\right]$$

$$\phi(t) = 0$$

Equation 4.29

As with the analytic pulses from the first section, the retrieved FROG trace of the Lorentzian spectral field is nearly indistinguishable from the input trace. This can be seen in the top panels of Figure 4.19. The input and output temporal magnitudes and phases

are also nearly indistinguishable. Although the method for producing the electric field of the pulse was somewhat different than for the first analytic pulses, it is not surprising that there would be good agreement between the input and output phases and magnitudes. It is not surprising because after the spectral field is inverse Fourier transformed to the time domain, the FROG traces were formed using the same procedures for the analytic pulses detailed in the first section. This method produces a FROG trace that obeys the time-bandwidth product relation.

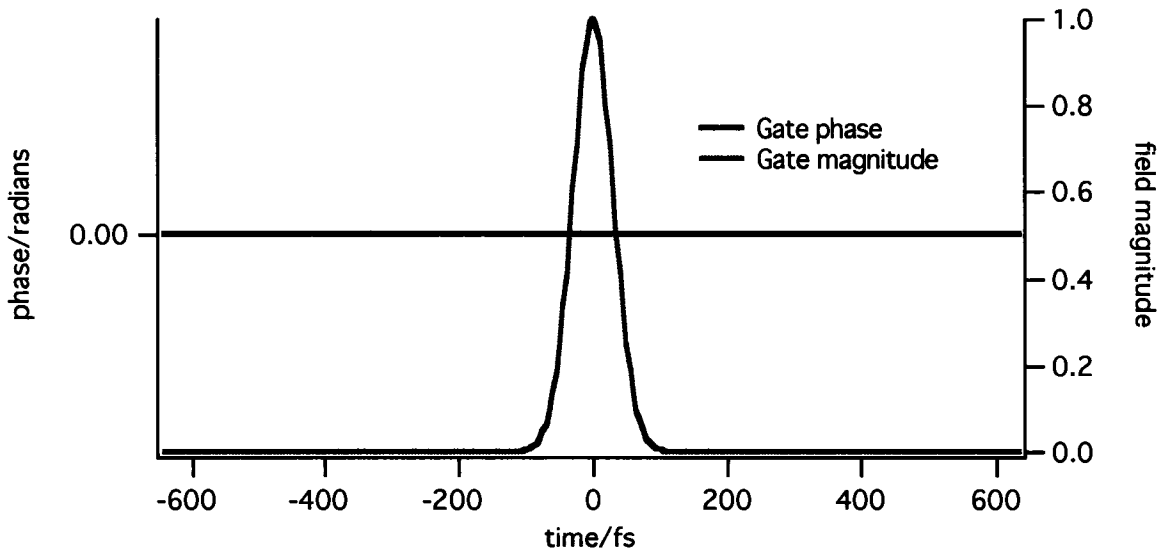


Figure 4.18 The temporal phase and magnitude of the gate pulse used to simulate the FROG traces of the model single molecule and distribution of molecules fluorescence FROG traces.

This simulational example can be viewed as a model for the emission from a single gas phase dye molecule with a fluorescent lifetime of ~4/0.011 fs^{-1} (from Equation 4.25) or ~ 360 fs full width half max. While this lifetime is much shorter than that of the

dyes typically used in time resolved fluorescence Stokes shift experiments, it provides a starting point for constructing an ensample (or distribution) of emitters in solution.

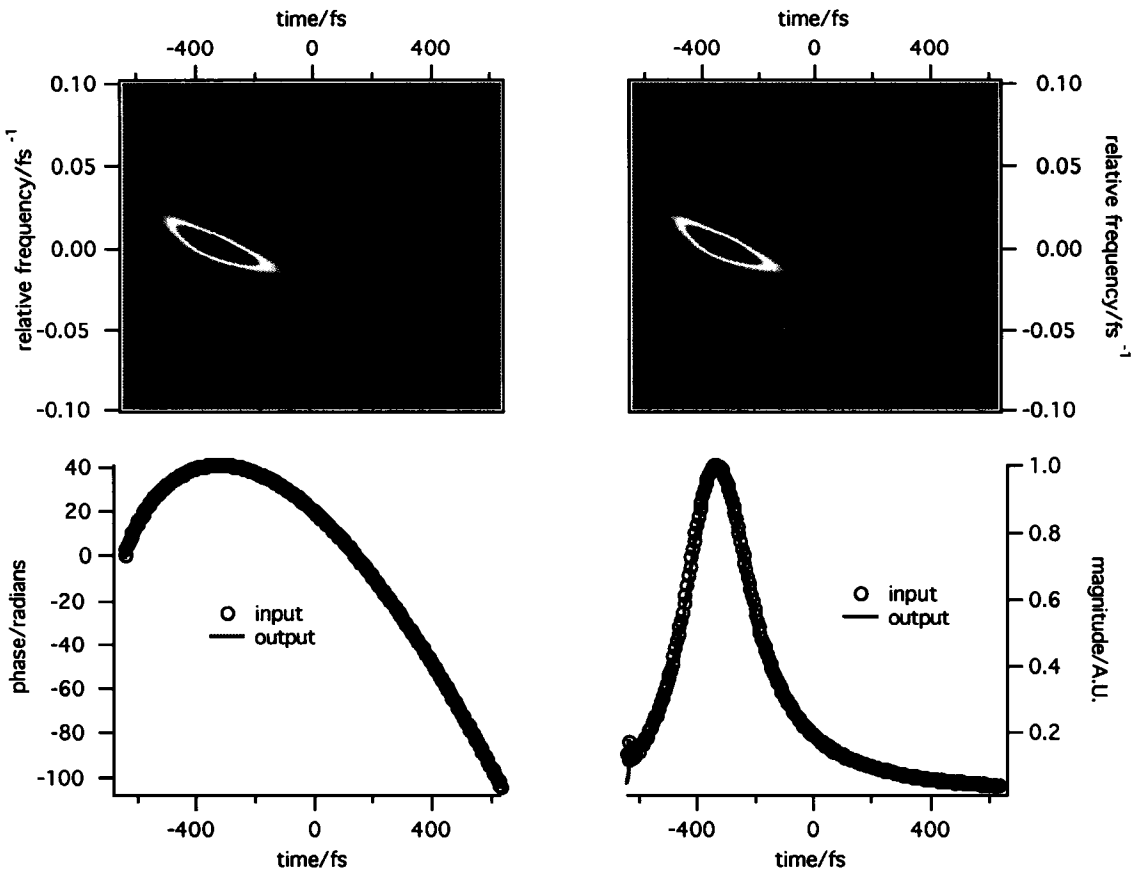


Figure 4.19 Clockwise starting with the upper left corner: Simulated FROG trace created using the Lorentzian spectral field described by Equation 4.24 and Equation 4.27 and the gate pulse described by Equation 4.29, the retrieved FROG trace, the temporal magnitudes of the simulated (input) and retrieved (output) fields, the temporal phases of the simulated and retrieved fields.

Construction of a Simple Distribution of Emitters from Lorentzian Spectral Fields

To model a distribution of emitters three Lorentzian spectral fields were constructed with slightly different central frequencies. The spectra from these spectral fields are shown in Figure 4.20 and labeled Field 1, 2 and 3 in order of decreasing central frequency. Field 2 is centered at 0 fs^{-1}, while Fields 1 and 3 are +0.01 and –0.01 fs^{-1}

respectively. Field 2 is also identical to the field from the previous simulated FROG trace shown in Figure 4.19.

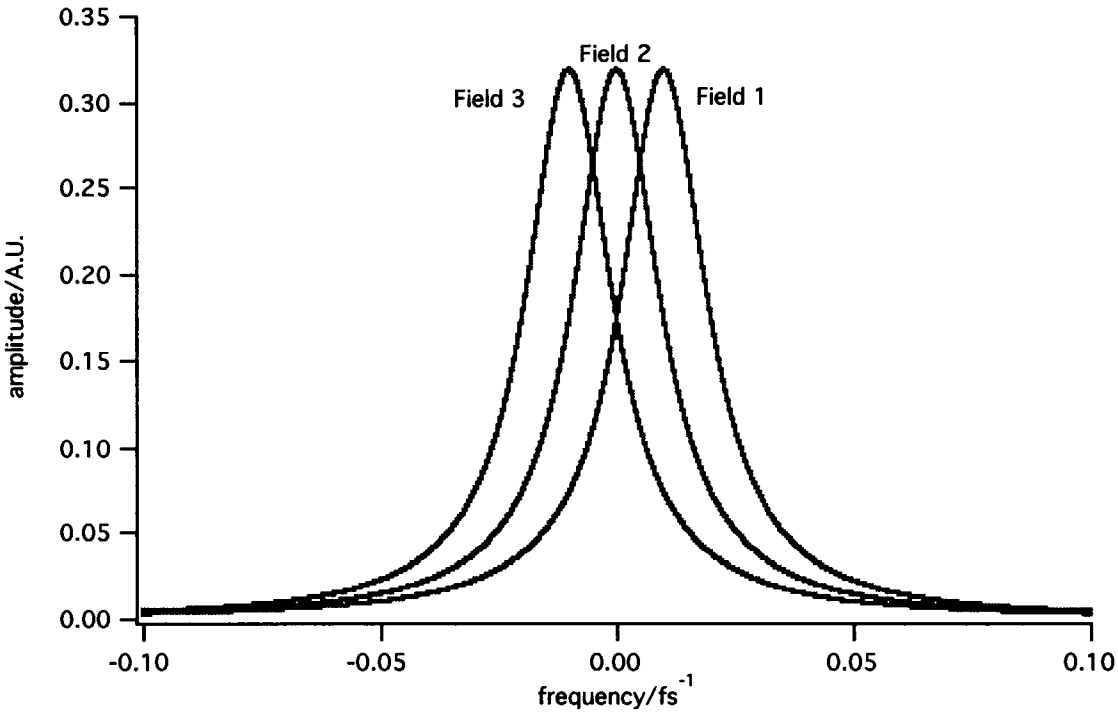


Figure 4.20 Spectra of the three Lorentzian spectral fields used to construct a distribution of emitters.

Instead of constructing a total electric field by summing the fields, a total FROG trace was constructed by calculating FROG traces for each field individually and then summing the FROG traces. In an analogous procedure to the one described above, the Lorentzian spectral fields were individually inverse Fourier transformed to produce time domain electric fields. Then the FROG traces for each field were calculated using Equation 4.1, Equation 4.3 and Equation 4.4 to produce $I_{FROG}(\nu,t)_i$ for each of the fields denoted by the index i. The traces were then summed:

$$I_{FROG}(\nu,t)_{\text{total}} = \sum_i I_{FROG}(\nu,t)_i$$

Equation 4.30

The calculated FROG traces from each of the individual spectral fields and the summed trace can be seen in Figure 4.21. Summing the FROG traces instead of summing the fields directly simulates the incoherent addition of the electric fields of individual emitters in the sum frequency gating step of the experiment. The Gaussian pulse shown in Figure 4.18 was used as the gate in the simulated FROG traces of all three Lorentzian spectral fields pictured in Figure 4.21.

Figure 4.22 shows the total FROG trace and the results of the phase retrieval. The upper right panel shows the retrieved FROG trace exhibiting the same modulation that can be seen in the retrieved traces of the simulated fluorescence signals shown in Figure 4.7 Figure 4.9, Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.14. Since the modulation is not observed in the retrieved field shown in Figure 4.19 we can reasonably conclude that the modulation is indeed due to the attempt of the algorithm to match the bandwidth of the input (simulated) FROG trace by creating rapid modulation in the field amplitude. In essence, the algorithm has to generate its guess for the electric field as a series of temporally short pulses in order to match the input bandwidth of FROG traces created by incoherent addition of signals. Following this rational, the bandwidth of the FROG trace will determine the amount of modulation. So, FROG traces with larger bandwidth such as those created with a Stokes shifting Gaussian spectral profile (Figure 4.7 Figure 4.9, Figure 4.10, Figure 4.11, Figure 4.12 and Figure 4.14) will require more rapid and higher amplitude modulation of the field amplitude. The plot of the output magnitude shown in the lower right panel of Figure 4.22 still shows significant evidence of the original (or input) magnitude. In fact the modulation appears to sit on top of a temporal envelope that resembles the input. This is in contrast to the magnitude plot of

the retrieved field in Figure 4.10, which appears to have no such underlying envelope. It could be concluded that this difference is due to the higher bandwidth of the simulated FROG trace.

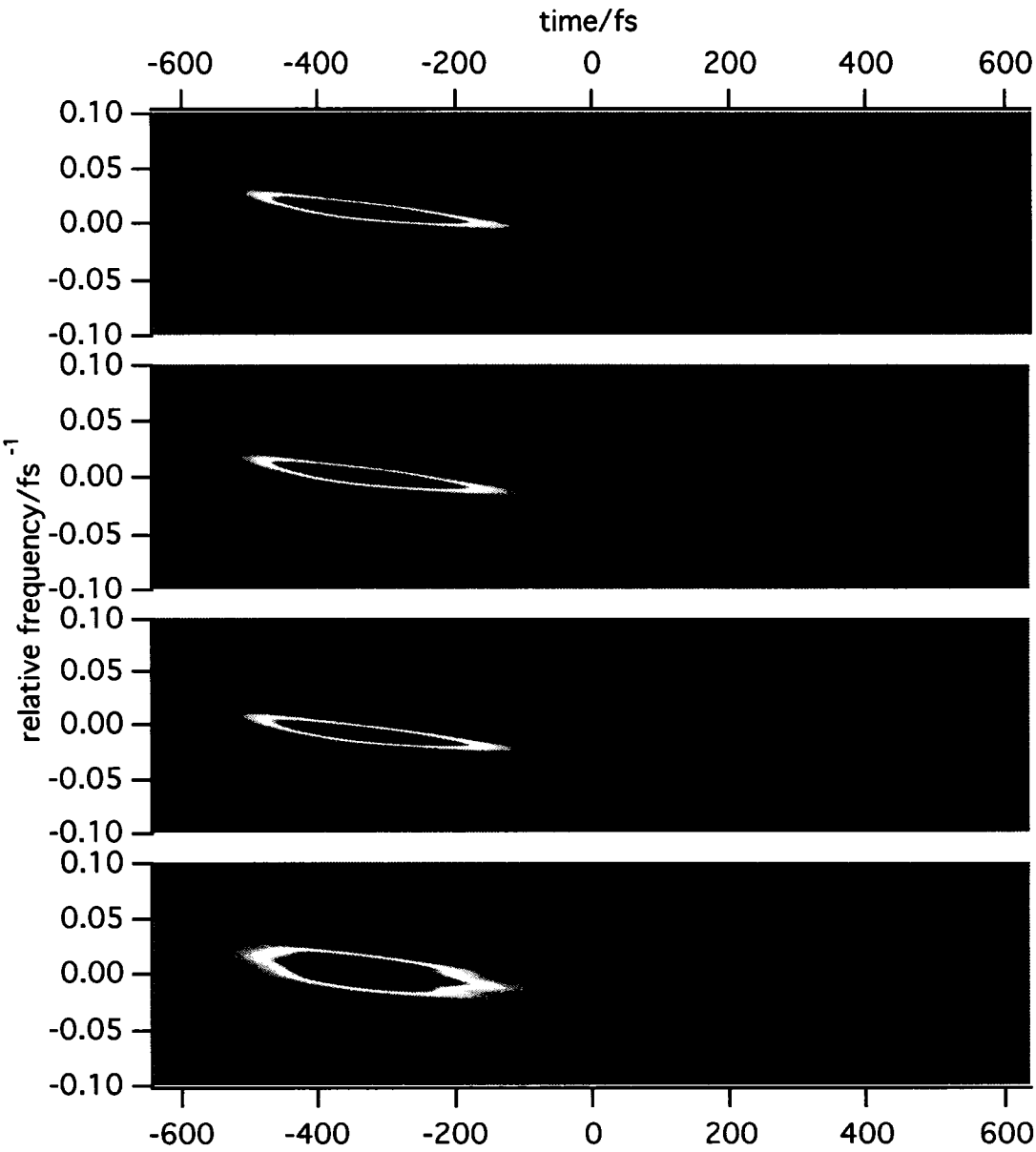


Figure 4.21 Simulated FROG traces calculated from each of three Lorentzian spectral fields labeled Field 1, Field 2 and Field 3, and the total FROG trace calculated by summing the three calculated traced labeled Total FROG.

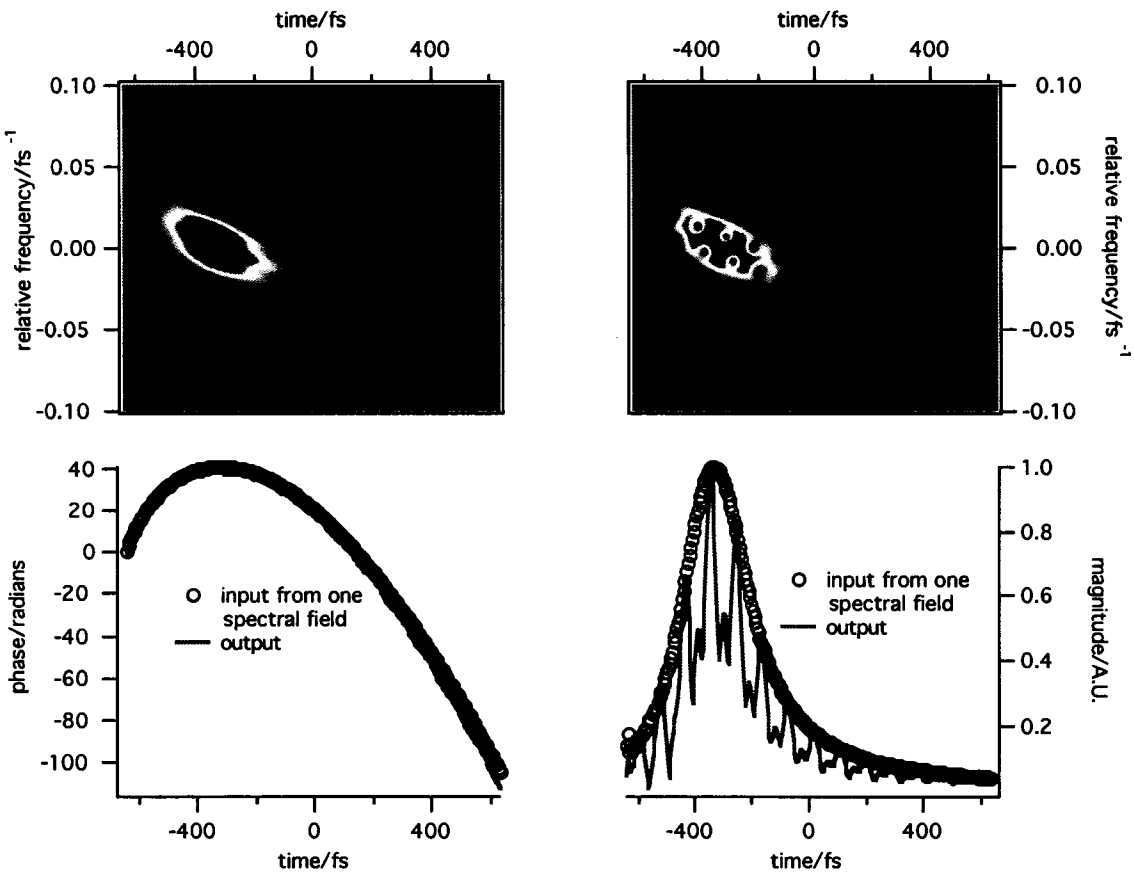


Figure 4.22 Clockwise starting with the upper left corner: Simulated FROG trace created using the sum of Lorentzian spectral fields FROG traces described by Equation 4.24, and Equation 4.27 with gate pulse described by Equation 4.29; the retrieved FROG trace; the temporal magnitudes of one of the simulated, summed spectral fields (labeled input) and retrieved (output) fields; the temporal phases of one of the simulated, summed spectral fields and retrieved fields.

The Uniqueness of Retrieved Fields from Analytic Pulses and Distributions of Emitters

The phase retrieval algorithm searches parameter space for an electric field that satisfies the constraints of both the measured data and of the experimental conditions that give rise to the FROG signal, which also know as the mathematical constraint. The algorithm searches parameter space by making projections from one constraint to the other.

The mathematical constraint is represented by the form of the function for calculating the FROG signal. For this work the form of the FROG signal is the so-called outer product of the gate pulse field and the trial pulse field, Equation 4.1. It accounts for the sum frequency process by which the gated fluorescence signal is gated by the gate pulse.

The data constraint is mathematically represented in the algorithm by the substitution of the square root of the input FROG trace, $\sqrt{I(\omega,\tau)}$, for the magnitude of the calculated FROG trace of the current iteration, $S_s(\omega,\tau)$.

$$S_s(\omega,\tau)^{\dagger} = \sqrt{I(\omega,\tau)}\frac{S_s(\omega,\tau)}{|S_s(\omega,\tau)|}$$

Equation 4.31

This results in a new calculated FROG signal, $S_s(\omega,\tau)^{\dagger}$, whose magnitude is derived experimentally and whose phase is determined by the algorithm.

Figure 4.23 shows the path taken by the phase retrieval algorithm to reach the solution that satisfies both the mathematical and data constraints for an input FROG trace of an analytic pulse. The curves in the diagram represent the boundaries of the parameter space populated by trial FROG signals, $S_s(\omega,\tau)$, that satisfy each of the respective constraints. The straight lines represent the projections of the algorithm between the parameter space regions that satisfy each of the constraints. The initial guess for the electric field of the undetermined pulse or fluorescence generates a FROG signal by being gated according to Equation 4.1, Equation 4.3, and Equation 4.4. The subsequent application of the data constraint moves the trial FROG signal the shortest distance into the parameter space populated by parameters that satisfy the data constraint. Repeated

alternate applications of the data and mathematical constraints cause the algorithm to move along the boundaries of the two constraint regions until it reaches the point at which the regions intersect. The intersection represents the parameter set in the trial FROG signal parameter space that satisfies both constraints. The trial electric field that produced the trial FROG trace that satisfies the constraints is therefore the electric field of the undetermined pulse.

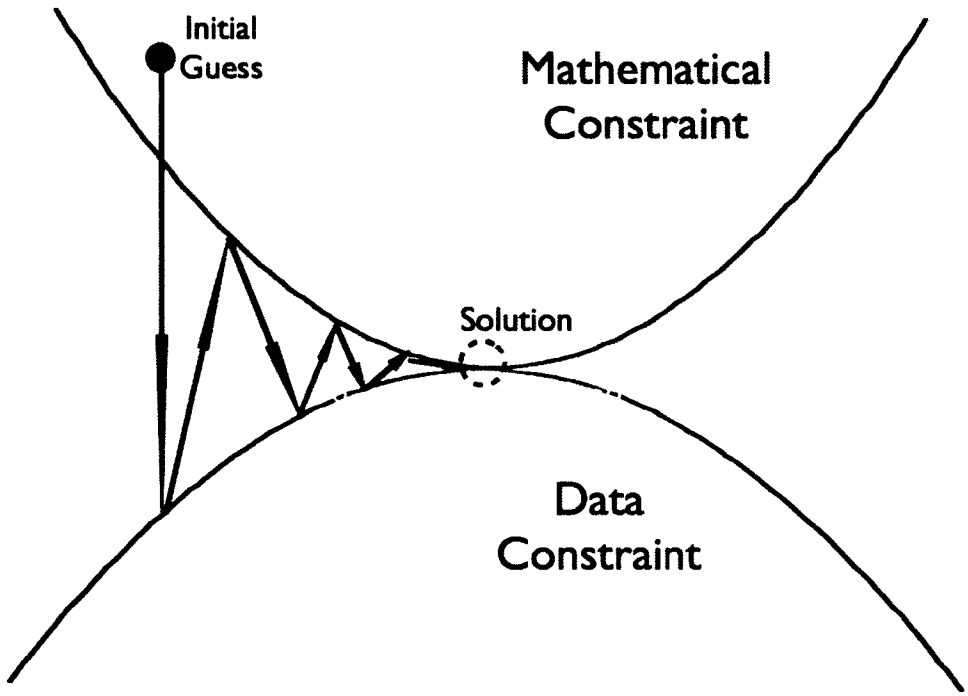


Figure 4.23 A schematic of the path that the phase retrieval algorithm takes through the parameter space searching for the electric field that satisfies both the mathematical and data constraints. The algorithm starts from an initial guess for the electric field and then makes alternating projections to the data and mathematical constraints. For analytical signals, the curves intersect at the unique solution that satisfies both constraints.

The retrievals of fields that are not adequately described by Equation 4.2 or whose FROG signals are not completely described by Equation 4.1 will not have a

unique solution that satisfies both constraints, depicted below in Figure 4.24. Instead of converging to a single point, the algorithm will become trapped in an area between the two regions at their locally closest point. Furthermore, the field produced by the algorithm after an arbitrary number of iterations will depend on the initial guess because there is no unique intersection of the constraints.

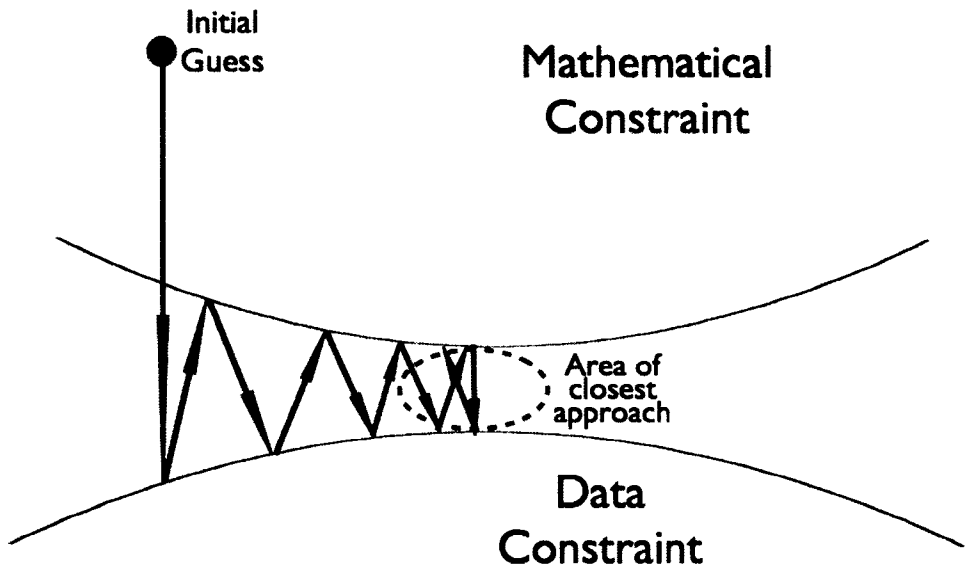


Figure 4.24 A schematic of the path that the phase retrieval algorithm takes through the parameter space searching for the electric field that satisfies both the mathematical and data constraints. The algorithm starts from an initial guess for the electric field and then makes alternating projections to the data and mathematical constraints. For signals from a distribution of emitters, the curves will not intersect at a unique solution that satisfies both constraints.

To verify this assertion, retrievals of the same simulated input FROG trace were performed using three different initial guesses, or seed pulses, for 10,000 iterations of the algorithm. The input FROG trace used was the trace pictured in Figure 4.10. Three

different seed pulses were created using the procedure described to produce the seed pulse pictured in Figure 4.2. The magnitude of the pulse is Gaussian, but each point of the phase is a random number between –1 and 1. Although the magnitudes for each pulse were the same, the different phases place the seeds in different regions of the mathematical constraint parameter space. Different starting points will cause the algorithm to follow a different path to the area of closest approach and after an equal number of iterations produce a different electric field.

The best demonstration of the difference of the retrieved fields produced by starting with different seed pulses can be seen in the plots of the retrieved magnitudes in Figure 4.25. Clearly, the magnitude plots of results from three different seed pulses are different. Further proof of this is seen in the Fourier transforms of the retrieved magnitudes shown in Figure 4.26.

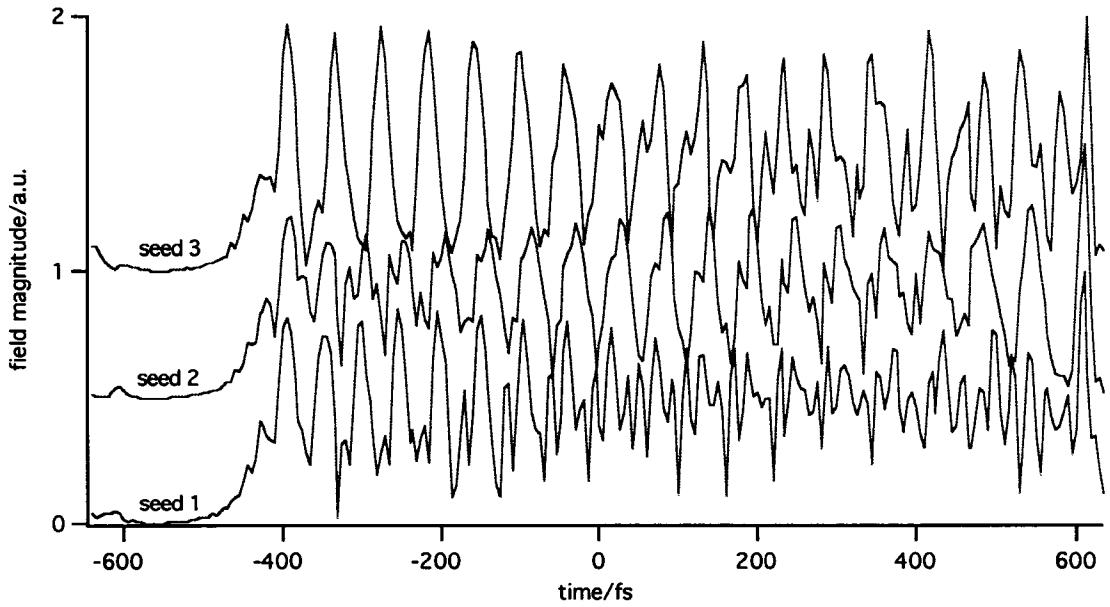


Figure 4.25 The retrieved magnitudes of the fields produced from three different seed pulses. The input FROG trace is pictured in the upper left panel of Figure 4.10. The three seeds were produced using the same Gaussian magnitudes, but each having random phase.

The Fourier transforms all show some structure around 0.018 fs^{-1}. This is most likely related to the bandwidth of the input FROG trace. As discussed in the previous section, the phase retrieval algorithm will attempt to reproduce the bandwidth of the input FROG trace by modulating the magnitude of the trial electric field. Because the phase retrieval algorithm assumes the time-bandwidth product relation, the modulation of the electric field will be of higher frequency for wider bandwidth FROG traces.

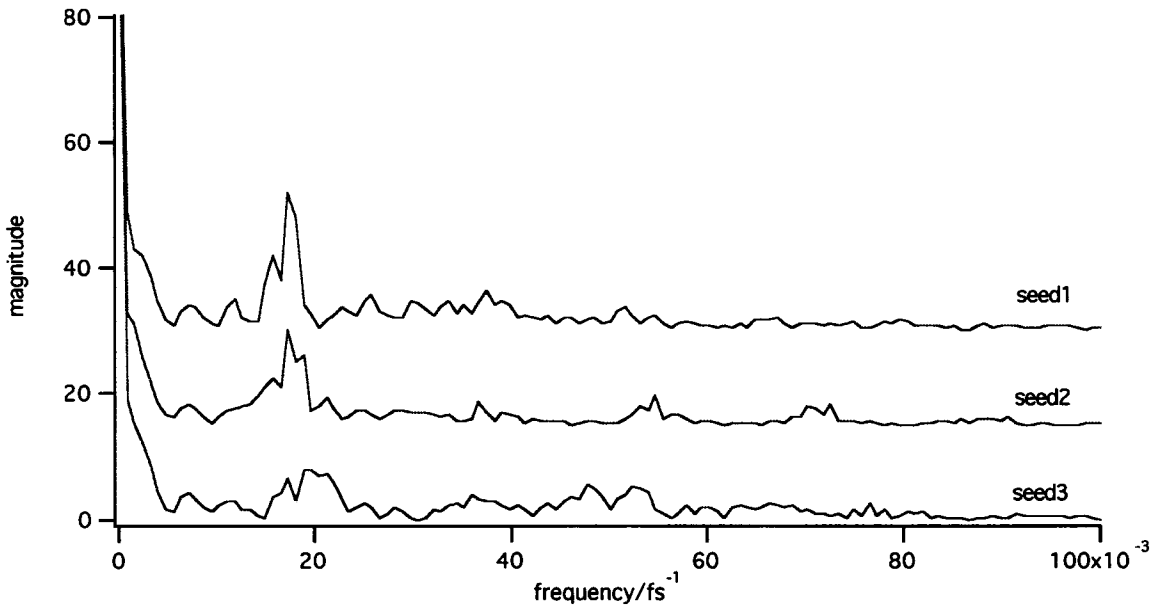


Figure 4.26 The Fourier transform of the retrieved magnitudes pictured in Figure 4.25.

Although the structure of the retrieved magnitudes resulting from each of the seed pulses look different, the retrieved phases shown in Figure 4.27 show some similarities and are similar to the integrated Stokes shift. The similarity of the retrieved phases to each other and to the integrated Stokes shift indicates that the area of closest approach is populated by parameter sets that have potentially relevant phase information. Drawing from the similarity of these retrieved phases and generalizing the results of the retrievals in previous sections, we can reason that the accuracy and reproducablity of retrieved

phases will be higher for input traces that have a more rapid Stokes shift and/or have lower bandwidth.

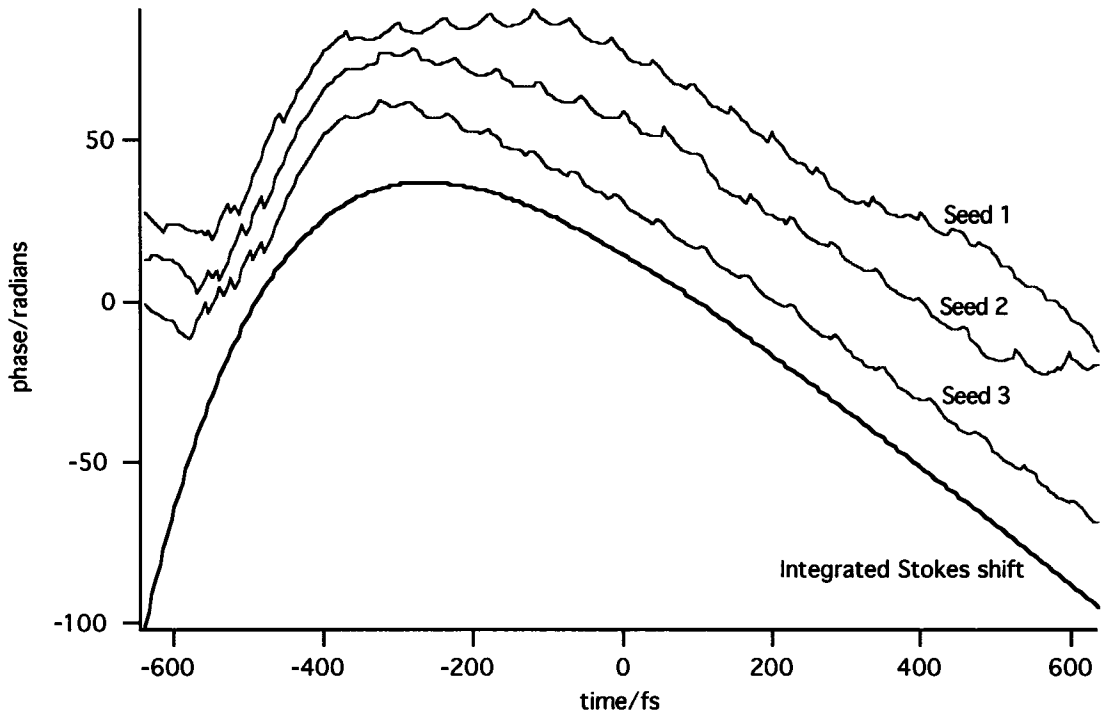


Figure 4.27 Retrieved phase of the simulated FROG trace pictured in the upper left panel of Figure 4.10 using three different seed pulses as well as the integrated Stokes shift.

In conclusion, the phase retrieval algorithm constructed for this work has been shown to properly retrieve the electric fields from simulated FROG traces that have been constructed from pulses that can be described by Equation 4.2. It was also shown that the phase retrieval algorithm fails to retrieve the correct/complete field for FROG traces constructed with signals that violated the time-bandwidth relation, which were constructed to resemble time resolved fluorescence emission from solvated dye molecules. FROG traces that are constructed as incoherent sums of other FROG traces are were also shown to cause the algorithm to return incomplete fields. However, while the phase retrieval algorithm fails to retrieve the complete electric fields of signals

produced in this manner, the retrieved phase bares a striking resemblance to the phase of the input signal. It was also shown that a windowed rolling average of the time derivative of the phase resembles the Stokes shift of simulated FROG traces constructed to resemble time resolved fluorescence spectra.

Chapter 5

Discussion of Results

Choice of Spectral Fitting Function

Using a sum of Gaussians to fit the time resolved fluorescence spectra was advantageous because it allowed for the possibility of uncorrelated spectral diffusion of different portions of the spectra. Moreover, it fit the features of the measured spectra better than other functions such as the lognormal.

It also appears that that a sum of Gaussians may have been a better fit for the spectra from the work of other groups. For instance Horng et al[10] page 17317, figure 5, panel B shows lognormal fits for several time resolved fluorescence spectra of C153 in dimethalsulfoxide (DMSO). The earliest time spectrum exhibits (in the figure reproduced below) a substantial shoulder on the red edge that cannot be reproduced using the lognormal function. Also from this source, the lognormal also seems to over estimate the intensity of the red tail.

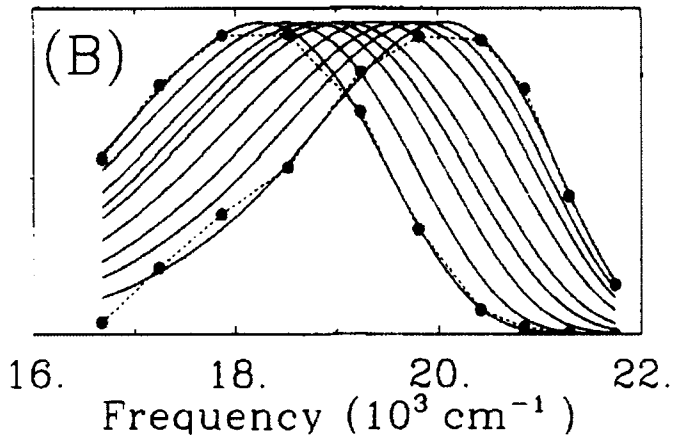


Figure 5.1 Reproduction of the time resolved fluorescence spectra of C153 in DMSO with lognormal fits (solid lines) and measured data point (dots), from figure 5, panel B of Horng et al.[10] Spectra shown correspond to 0, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, and 50 ps.

The early C153/DMSO spectra from Horng et al. compare with the early spectra of C153/acetonitrile at crystal angle 59° shown in Figure 3.2, page 37. The 450 and 500 fs spectra from the acetonitrile data show evidence of a shoulder on the red edge of each spectrum that is shifted to the red of the peak maximum by about 30 THz, or 1000 cm^{-1}. The spacing is similar to the vibronic spacing exhibited by C153 fluorescence in non-polar polar solvents such as 2-methylbutane and cyclohexane of 1000-1200 cm^{-1}.[10, 32] The steady state fluorescence spectra in acetonitrile and methanol also show some evidence of the underlying vibronic structure. The steady state fluorescence of C153 in acetonitrile can be seen in Figure 1.3 and in methanol Figure 5.2 for methanol.

Other than loss of direct comparison with a de facto standard of the lognormal function, there is no loss of any dynamical information by adopting a sum of Gaussians function form. While the sum of Gaussians function lacks an asymmetry parameter, the dynamical information it expresses can be assessed by considering the relative positions of the two Gaussian peaks as well as their respective amplitudes.

Limited Bandwidth and Dynamical Range

Limitations of the broadband detection technique are related to the bandwidth throughput of the sum frequency generation process. The sum frequency signal is produced by a nonlinear optical process in a nonlinear optical (NLO) crystal, specifically a beta-barium borate crystal in this instrument. The efficiency of the process is governed by the phase matching of the two input beams; that is, fluorescence from the dye solution and the gate pulse, within the NLO crystal. Phase matching is the condition where the phases of the two propagating pulses are matched with the periodic variation of electron density or polarizability along their propagation vectors within the NLO crystal. The degree of phase matching will vary with the frequency of the two pulses as well as the direction the pulses travel within the crystal. As the propagation vectors of the two input pulses deviate from the optimal direction within the crystal, either due to the frequencies of the input pulses or to geometry, the efficiency of the nonlinear optical process will decrease. This decrease in sum frequency generation efficiency results in a decrease in the bandwidth of the detected upconversion signal.

If the effects of phase matching within the NLO crystal are disregarded, the SFG signal as a function of frequency (the SFG spectrum) would be the convolution of the spectra of the two input pulses and would represent the ideal upconversion spectrum. In this case the ideal upconverted spectrum would be almost indistinguishable from the real time resolved fluorescence spectrum because the spectral width of the gate pulse is approximately 15% narrower than that of the real spectrum. The convolution of these two spectra should produce an ideal upconverted spectrum that is less than 1% wider than the

original fluorescence spectrum. The plot in Figure 5.2 shows the steady state fluorescence spectrum of C153 in methanol in black and the upconverted fluorescence spectrum 9 ps after excitation in red. The steady state spectrum was produced using the same sample, sample flow cell and excitation source as the upconverted spectrum. However, instead of gating the fluorescence it was captured by an optical fiber connected to a spectrometer.

Because the steady-state emission spectrum is integrated over all time and the emission spectrum shifts in time, one might anticipate that the steady-state spectrum does not truly reflect the spectrum after all the shifting is over. If we had access to time correlated single photon counting instrumentation, we might be able to measure the spectrum that arises after all the Stokes shifting is over. However, the steady state spectrum does provide an estimate of the lineshape and linewidth. This is a reasonable estimation because the vast majority of the spectral diffusion due to polar solvation dynamics in the systems studied occurs within the first 500 ps after excitation.[10] This represents only 15% of the total integrated emission intensity for the 3 ns fluorescence lifetime of C153.

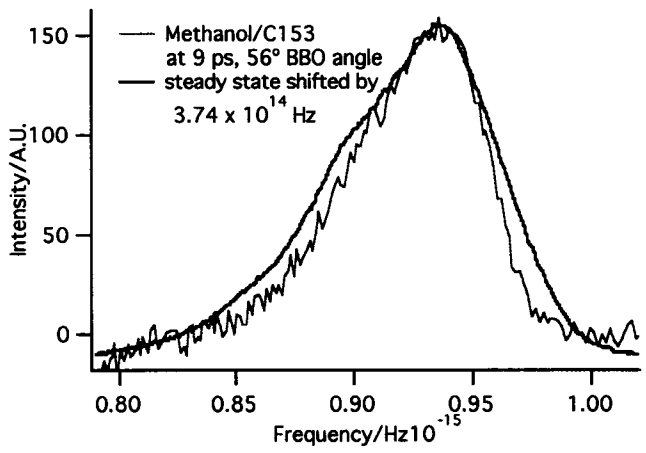


Figure 5.2 The SFG spectrum recovered 9 ps after excitation (red trace) and the steady state fluorescence (black trace) for C153 in methanol.

While we would predict the SFG spectrum to be broader than the original steady-state spectrum, it is clear from Figure 5.2 that the SFG gated fluorescence is narrower than the steady state spectrum. This narrowing is most likely due to the reduced SFG efficiency because the gate pulse and fluorescence cannot achieve sufficient phase matching across their entire spectral widths. In this case the phase matching criteria of the SFG process acts as a frequency dependent filter. To quantify this filtering effect the full width at half maximum (FWHM) of the steady state and upconverted spectra can be measured. The FWHM of the steady state signal is 8.13×10^{13} Hz, while the FWHM of the upconverted spectrum is 6.65×10^{13} Hz. Because the steady state spectrum should be at least nominally wider than the real time resolved spectrum at 9 ps, the FWHM throughput for this upconverted spectrum of approximately 82% probably represents a lower limit.

The bandwidth throughput limitation was somewhat alleviated was to average the upconverted spectra obtained by collecting spectra at several NLO crystal angles. By changing the angle of the NLO crystal with respect to the propagation direction of the input beams, different portions of the fluorescence spectrum can achieve optimal phase matching. The results of this procedure can be seen in Figure 5.3.

The red trace in the left panel is the upconverted spectrum 9 ps after excitation and is designated with a label of 56°. The value of 56° was the reading from the rotational mount of the NLO crystal and only has quantitative meaning in relation to the other crystal angles and not to the propagation direction of the input beams. The green and blue traces are the result of changing the crystal angle by one and two degrees and are

designated by 57° and 58°, respectively. In this instrument, rotating the NLO crystal to higher angular readings favored phase matching of the higher frequencies. This effect can also be seen in the left panel of Figure 5.3 as the peak frequency of the 9 ps spectrum shifts to the blue when the crystal is tuned to higher angles.

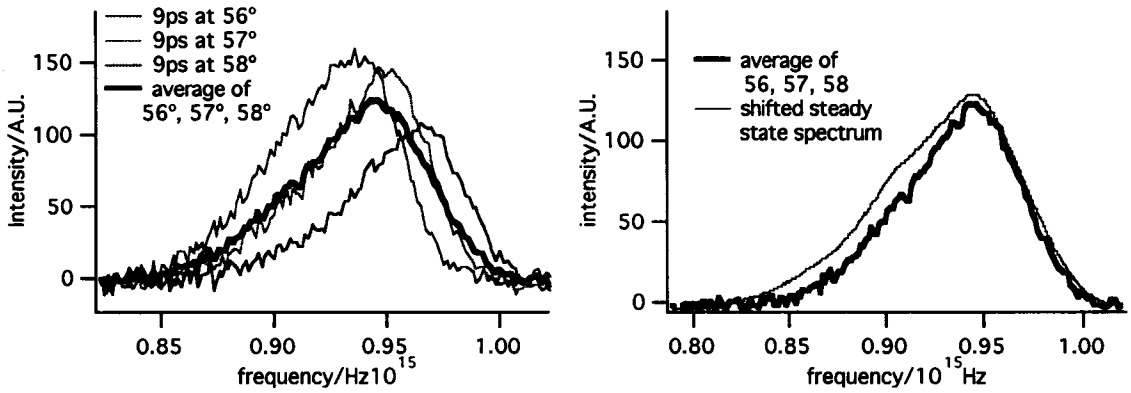


Figure 5.3 (Left) SFG spectra of Coumarin 153 in methanol collected for varying nonlinear optical crystal angles (colored traces) and average of the traces (black trace). (Right) Comparison of average SFG spectra with the steady-state spectrum of C153 in methanol shifted by 3.82 x 10^{14} Hz.

The black trace in Figure 5.3 was produced by averaging the equally weighted upconversion spectra collected at the three angles 56, 57, and 58°. The average of the three angles is plotted in the right panel of Figure 5.3 along with the steady state spectrum to demonstrate the improvement in the replication of the blue edge of the steady state spectrum after averaging over three crystal angles. Therefore, while the broadband detection technique is limited by total bandwidth throughput, this limitation can be largely overcome by averaging over a small number of crystal angles.

Clearly, performing broadband fluorescence upconversion experiments with a limited range of sum frequency crystal angles limits the total bandwidth throughput of the technique. As a consequence, the total detectable dynamic Stokes shift of the experiment

is also limited. The total dynamic Stokes shift recorded over 8 ps for Coumarin 153 in acetonitrile at the crystal angle of 59° is shown in Figure 3.3. The total Stokes shift is 9.3 THz, or 310 cm^{-1}, for the time resolved fluorescence Stokes shift experiment performed at this angle, while the total shifts for angles 58° and 57° are 6.9 THz (230 cm^{-1}) and 4.8 THz (160 cm^{-1}), respectively. While the 59° single-angle experiment showed the largest shift it was still only about 15% of the total shift observed by Horng et al[10] of 2.28 x 10^3 cm^{-1}. This shift was measured from fluorescence upconversion experiments using narrow bandwidth detection methods, i.e. measuring ten to fifteen fluorescence decays detected with a photomultiplier tube at several corresponding crystal angles. A total Stokes shift from the combination of 59° and 57° single-angle experiments yields a total combined Stokes shift of 40 THz (1.3 x 10^3 cm^{-1}). This combined shift was calculated by subtracting the highest frequency of the 59° experiment and the lowest frequency of the 57° experiment. Remarkably, this bandwidth limitation does not prevent the broadband method from quantitatively reproducing the gross relaxation dynamics expressed in the solvent response function *C(t)*.

Reproduction of Gross Solvent Relaxation Features Using Single Angle Measurements

As mentioned previously, the effective dynamical sensitivity of the broadband upconversion experiment is somewhat limited. However, the technique does recover the proper relaxation dynamics represented by the solvent response function *C(t)*.

Ideally, the solvent response function is characterized only by the solvent and not by the probe molecule or experimental technique used to measure it. Because the solvent

response Equation 3.4 is normalized by the total Stokes shift observed in the experiment it should also be independent of any experimental conditions that might pose moderate limitations on the total shift. This appears to be the case with the single-angle broadband experiment.

Figure 3.4 (page 40) shows the solvent response function, *C(t)*, calculated from both the average frequency Equation 3.2 and peak frequency for acetonitrile at crystal angles 59°, 58°, and 57° along with their respective fits to a bi-exponential function, Equation 3.5. The results of the fits are tabulated in Table 3.1 along with the results of Horng et al labeled as Lit. 1. Horng and coworkers report the exponential fits of *C(t)*, which were calculated as the average of solvent response functions obtained averaging *C(t)* based on both the peak and first moment frequencies.

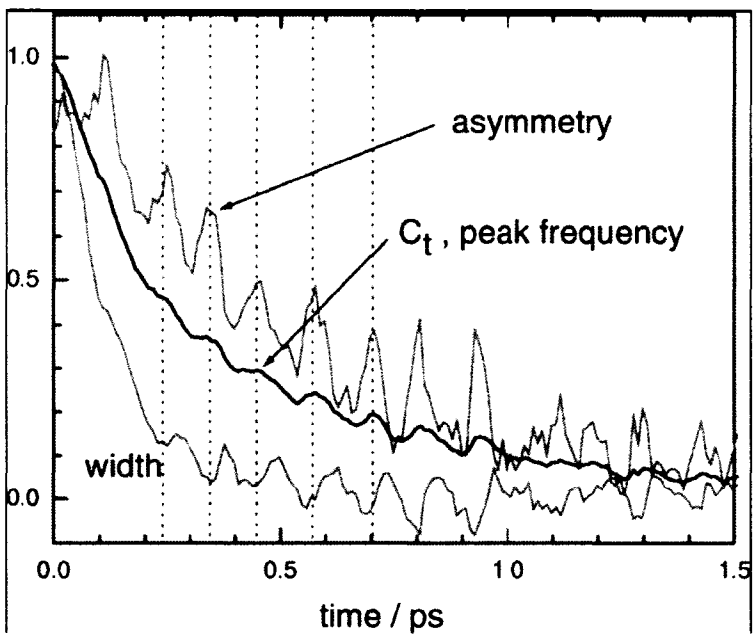


Figure 5.4 Coumarin 153/acetonitrile normalized solvent response function C_t, asymmetry, and width measured by a similar, but more sophisticated broadband fluorescence upconversion experiment.[33]

Figure 5.4 shows the results from a similar broadband fluorescence upconversion experiment performed by Zhao and coworkers studying Coumarin 153 in acetonitrile.[33]

Although this group does not report fitting their solvent response function, a reasonable estimate for the 1/e time of approximately 400 fs can be made directly from the graph. This 1/e time compares very favorably with that of the 59° single-angle experiment of 425 ± 25 fs.

The discrepancy between the exponential time constants of the response functions produced from the average and peak frequencies (Table 3.1 and Table 3.2, page 40) points to the possibility that important dynamical information is contained in both terms of the sum of Gaussians fit function. It seems that the average frequency is more representative of the average free energy dissipation in the solute/solvent system. In addition, the RMS of the noise in the tail section of solvent response functions calculated from the average frequency is noticeably lower than those calculated from the peak frequency (see Figure 3.4, page 40). This is not too surprising considering that the peak average is determined from the entire spectrum, but the peak frequency is largely determined from a subset the spectrum. Therefore, the peak frequency will inherently be more sensitive to noise.

Again from Figure 3.4, the response functions measured at 57° and 59° are remarkably similar considering that they are nominally tuned to different parts of the fluorescence spectrum. The ratio of the faster (τ_1) and slower (τ_2) time constants is nearly identical. It can also be seen that the 57° and 59° response functions appear to level out (at least within the noise) after about 3000 fs, while the 58° curve still has negative slope. The presence of the slope is confirmed by its time constants that are nearly three times that of the other two.

The second solvent studied in this manor was methanol whose average frequency solvent response functions for crystal angles 56°, 57° and 58° are shown in Figure 3.5 (page 41) with associated fit parameters in Table 3.3 (page 42). The difference in solvent relaxation rates from that of acetonitrile is quite evident from the long sloping tails of response functions and from the additional 7-9 ps time constant needed to fit the response function. The longer relaxation times are largely due to the presence of hydrogen bonding in methanol. The hydrogen bonds that exist between methanol molecules hinder its reaction to the dipole of the solute and thus slow the relaxation somewhat.

In similar fashion to the acetonitrile data, the 56° and 58° fit parameters are in good agreement, with the middle angle 57° as the outlier. In contrast to the acetonitrile data, the agreement of the time constants with the literature values presented in the table is not as good. The fastest time constant is in good agreement, but there is no close match for the remaining two. The literature source reports a time constant of 280 fs that *should* be quite evident in the data given the time resolution and time range of the experiment.

However, a broader survey of the literature reveals solvent response functions obtained from time resolved fluorescence Stokes shift experiments with C153 with two and three time constants.[34] All sources report a fast constant of 30 to 75 fs, the so called inertial component. The next fastest component ranges from 550 fs to over 3 ps.

Most of these references including Horng et. al. used the "time zero fluorescence spectrum estimation" procedure developed by Chapman and Maroncelli to estimate the fluorescence spectrum immediately after the vertical transition of the dye from the S_0 to S_1 surfaces (Figure 1.1, page 4).[35] This estimation procedure involves deducing an equilibrium distribution function for the solute/polar solvent system from the absorption

and fluorescence lineshapes of the solute in a nonpolar solvent. Then, this distribution of solvent environments becomes the time zero distribution on the S_1 surface. This then allows for the estimation of the fluorescence spectrum before any solvent relaxation has occurred. Interestingly, the method presented here replicates the methanol and acetonitrile results pertaining to the fastest relaxation component without resorting to this estimation.

Reveling Fine Solvation Relaxation Features

Several groups of molecular dynamics (MD) researchers have predicted oscillations in the early time solvent responses based on results of their simulation of the polar solvation dynamics of acetonitrile, methanol and water.[8, 18, 21, 22, 36-40] The oscillation in the solvent response (see 110) are generally due to librational motion, which are small amplitude hindered rotational motions of solvents such as acetonitrile and methanol. It seems reasonable that the experimental methods described here should be able to confirm the existence of such features in the solvent response functions.

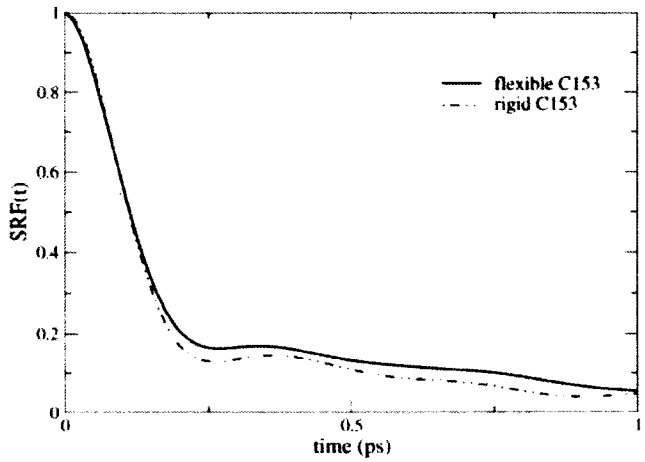


Figure 5.5 Solvent response function (SRF(t)) of C153 in acetonitrile calculated from MD simulations from Ingrosso et al, page 3562, figure 9.[22]

A closer view of the solvent response function of acetonitrile measured at 57°, 58° and 59° (Figure 3.8, page 44) shows evidence of the librational motion described in the molecular dynamics simulations mentioned above. Error bars associated with each plot are the sample deviations based on two replicate measurements for 59° and 58° data and three replicates for the 57° data. The first 500 fs of solvent response functions show oscillations larger than the error bars. Moreover, these oscillations appear at least somewhat correlated over that time period in all three curves, especially in the 59° and 57° curves. The period of these oscillations is 250-300 fs, which is in good agreement with the oscillations observed in the MD simulation attributed to librational motion.

The 58° data shows an additional feature (larger than the error bars) at 750 fs that is not readily identified in the other two data sets. The correlation of the oscillations between the three curves seems break down after approximately 600 fs, falling short of the observed time range in the MD studies. However, the amplitude of the oscillations observed in those studies decreases rapidly after the first 200 fs, so it is possible that it is present in the actual dynamics, but not resolved in the 57° and 59° experiments.

These same types of librational motions have been predicted by many of the same MD researchers for methanol.[36, 41] However, since the dipole moment of methanol is 1.7 D, compared with 4.1 D for acetonitrile,[36] the amplitude of the oscillations will be smaller for methanol. Indeed, only the 58° data for methanol shows evidence of the oscillatory behavior (Figure 3.9, page 44) and it is not as convincing as the acetonitrile evidence. The feature appearing at approximately 300 fs is similar to features seen by Kumar et al (Figure 5.6) [41] as well as Cichos et al for the same time period.[36] However, its amplitude compared with the associated error bars and the lack of evidence

in the other two angular data sets decreases the certainty of the comparison with the MD literature.

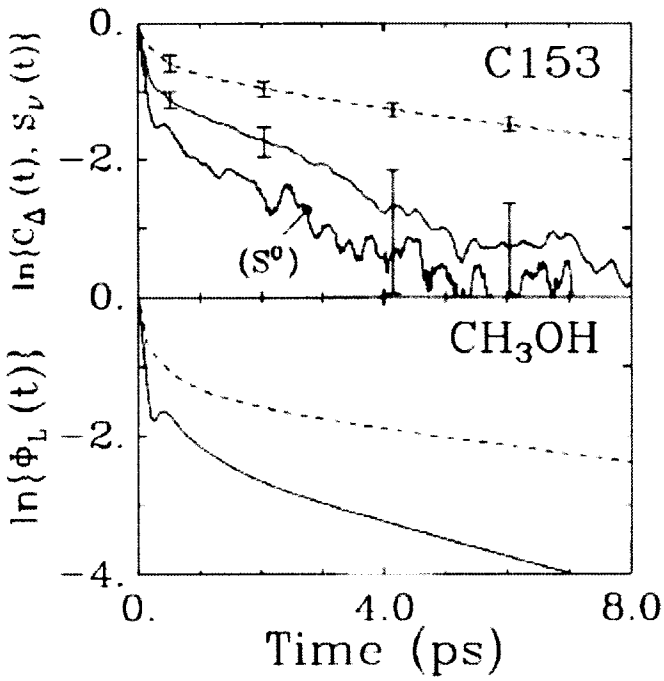


Figure 5.6 Molecular dynamics simulations of C153/methanol from Kumar et al showing the libraional motion of methanol at early times (top). The solid with error bars is the simulational C(t) resulting from a realistic model of C153. The solid curve marked (S^0) is the result from using an ionized single atom solute. The dashed line is an experimental result. The bottom panel shows a simulation using a model based on the bulk dielectric properties of methanol (solid line) and experimental results of a bulk dielectric measurement (dashed line).

The dynamics expressed in the separation of the peak positions of the individual Gaussians used to fit the acetonitrile fluorescence (Figure 3.6 and Figure 3.7, page 42) show the difference in spectral diffusion on the high and low frequency sides of the time resolved fluorescence. While the average frequency solvent response functions (Equation 3.4) for 59° and 57° show that the relaxation in acetonitrile is complete by about 3000 fs, the Gaussian peak position difference curves for 58° and 57° still shows evidence gradual

increase throughout the 8 ps range of the data. A similar feature to the 750 fs feature in the 58° average frequency solvent response function mentioned on the previous page is seen in the 58° peak position difference curve as a negative bump occurring at 750 fs. The positive feature in the average frequency solvent response function indicates an increase in the average frequency, while the negative bump in the difference curve indicates that the two Gaussian peak positions are closer. Taken together this indicates at the very least that the lower frequency portion of the spectrum experiences a blue shift that is greater than the blue shift experienced by the higher frequency portion.

As mentioned in previously in this chapter under the heading "Choice of Spectral Fitting Function" starting on page 100, the vibronic spacing of within the S_0 state of Coumarin 153 is 1000-1200 cm^{-1} depending on the solvent. If we make a rhetorical leap and assume that the higher and lower frequency Gaussians from the fitting function correspond to downward transitions terminating in the $v = 0$ and $v = 1$ vibrational states, respectively, then this could elucidate role of the dynamic intermolecular vibrational coupling of the solvent and solute during the relaxation process. The peak position differences exhibited in the data (Figure 3.6 and Figure 3.7) at the three crystal angles is somewhat shy of the 1000-1200 cm^{-1} spacing reported.[10] However, the bandwidth throughput of a single angle experiment probably decreases the magnitude of the peak position difference.

Another interesting behavior demonstrated by the peak position difference plots is the gentle upward slope that persists even after the average frequency solvent response function has stopped changing at approximately 2500-3000 fs (See Figure 3.4, page 40). Assuming that each Gaussian used in the sum of Gaussian fitting function represents a

transition from the vibrationally relaxed solute on the S_1 surface to a vibrational level ($\nu = 0$ or $\nu = 1$) on the S_0 surface, then the increase in peak frequency difference plotted in Figure 3.6 andFigure 3.7 (page 42) indicate a time dependent change in the energy spacing of these terminal vibration states. Obviously, this would not be due to any intrinsic change of the solute, but instead a change in the solvent environment in which it exists. More specifically, it points to a change in the vibrational coupling between the solvent and solute as solvent relaxation proceeds.

Zhao and coworkers[33] report oscillations with a period of approximately 100 fs in the lognormal fit parameters of C153/acetonitrile fluorescence shown in Figure 5.4 (page 107). They postulate that these oscillations, present during the first picosecond, are the result of low frequency C153 vibrational modes (previously observed in the gas phase[42]) coupling with intermolecular (collective) acetonitrile vibrational modes. Although, the sampling frequency of 50 fs in the acetonitrile data presented here is probably insufficient to resolve the 100 fs oscillations reported by Zhao, the comparison between the two studies demonstrates the plausibility of extracting transient vibrational coupling information from the type of high frequency and high temporal resolution fluorescence Stokes shift experiment.

Phase Retrieval

A further refinement of the broadband fluorescence upconversion experimental techniques comes from a more sophisticated approach to the data analysis that incorporates modern laser diagnostic measurements. Phase retrieval analysis allows for the deconvolution of the gate pulse from the upconverted fluorescence to provide

drastically increased time resolution and early time dynamical data free of the influence of the gate pulse.

The most direct comparison to the traditional analysis comes by calculating a solvent response function (Equation 3.13, page 50) defined using the windowed average frequency (Equation 3.12, page 50). The best example of a solvent response function calculated in this manner is from the fluorescence of C153 in acetonitrile shown in Figure 3.14, page 51. As mentioned in the previous chapter, the highly oscillatory behavior of the retrieved fluorescence electric field makes calculation of the time derivatives of the temporal phase challenging. Using the windowed average frequency, as defined in Equation 3.11, helps to alleviate this problem but still does not produce a smooth response curve. Much of the roughness stems from using a constant value for σ, a value related to the bandwidth of the fluorescence signal. The plot of the peak position differences (Figure 3.6, page 42) as well as the lognormal parameters reported by other groups[10, 33, 34] show that the width of the fluorescence signal increases sharply over the first picosecond. Manually varying σ reduces the roughness in different portions of the response curve. The value of 53 fs was chosen to minimize the roughness in as much of the relaxation as possible. Clearly, incorporating a variable σ into the calculation would be an improvement. This could be accomplished by measuring the width directly from the experimental data. Another strategy would be to use an optimization algorithm to minimize a roughness metric by adjusting σ at each time point.

The 1/e time for $C_p(t)$ of acetonitrile is between 425 and 500 fs, comparing well with the figure of 425 fs for the same data set analyzed using the sum of Gaussian method. It also compares favorably with the broadband fluorescence upconversion

literature source mentioned in the section named Reproduction of Gross Solvent Relaxation Features Using Single Angle Measurements on page 106 with a 1/e of approximately 400 fs.

For comparison, $C_p(t)$ was calculated for C153 in methanol at a crystal angle of 58° (Figure 3.16, page 52) and $\sigma = 57$ fs. This value was also chosen to minimize the oscillations in the curve. The roughness is even more problematic in this $C_p(t)$ than for that for the acetonitrile data. However, some of its features can still be correlated to the input data shown in Figure 3.15, page 52. The spike occurring at 200 fs in the solvent response function is due to the vertical band of upconversion signal that extends from the high frequency (top) edge of the spectrogram to almost the low frequency edge (bottom). This part of the signal could be excess upconverted pump pulse that is transmitted by the solution after it has been photobleached. It appears as a narrow spike because the bandwith of the signal is extremely wide at that point resulting in a very short coherence time. The 1/e time can be estimated to be over 1 ps from the graph.

One possible reason for the relative difference in quality of the calculated solvent response functions for acetonitrile and methanol is the increase in bandwidth of the fluorescence signal. Comparing the areas occupied by the upconverted fluorescence signals in Figure 3.10 (page 46) and Figure 3.15 (page 52) can demonstrate this, as do the Fourier transforms of the retrieved electric fields (Figure 3.17, page 53). By both of these measures, the methanol sample fluorescence is broader than that of the acetonitrile. The broader emission of the methanol sample could cause more noise on the edges of the spectrum where the upconversion efficiency begins to drop. Also, the increase in bandwidth results in a decrease in coherence time. This places the discontinuities in the

temporal phase closer together, which, in turn, makes determination of time derivative of the phase problematic.

Chapter 6

Conclusions

The inclusion of broadband detection in fluorescence upconversion instruments has proven to be a substantial improvement over the traditional method of narrow band detection. Fitting the time resolved fluorescence spectra of Coumarin 153 in polar solvents measured by broadband detection to a sum of Gaussians is also an improvement over the de facto standard lognormal function. Finally, reaching beyond treatment of the fluorescence signals measured in time resolved fluorescence Stokes shift experiments with steady state spectroscopic methods to incorporate analysis techniques of ultrafast/broadband optical physics shows promise in revealing details of solvation dynamics on the sub-100 fs time scale.

Moving beyond the lognormal lineshape description of the time resolved fluorescence signals from C153 in acetonitrile and methanol to the sum of Gaussians lineshape has revealed subtle features of the solvation dynamics of these two polar solvents that previous methods have been unable to resolve. Because the sum of Gaussians functional form has two peak frequencies it can more transparently track uncorrelated spectral diffusion in different parts of the time resolved spectrum. The sum of Gaussians also provides a more physical model of the fluorescence because each G

aussian is a representation of the downward transition from the S_1 surface to various vibrational levels on the S_0 surface. Furthermore, tracking the change in the peak frequency difference shows how the solute/solvent vibrational coupling changes over the course of the relaxation process, which might otherwise be missed if a lognormal is used.

In this study, results from just one single-angle broadband fluorescence upconversion experiment reproduced the gross dynamics of solvent response function measured by the traditional narrow bandwidth multiple-angle method for C153 in acetonitrile. This demonstrates the validity of the broadband approach for measuring solvation dynamics in very fast polar solvents such as acetonitrile. The gross features of solvation dynamics in methanol were also examined and found to be generally in agreement with other measurements from the literature, although the literature varied substantially.

The shear number of frequency points generated for each time step using broadband techniques allows spectral features to be characterized with high precision. The increase in precision over the traditional narrow bandwidth detection method makes it possible to resolve librational motion manifested as oscillations in the various solvent response functions and peak frequency difference curves. This librational motion was detected during the first picosecond of relaxation in the C153/acetonitrile system as oscillations of the average frequency solvent response function with a period of approximately 250 fs, which is in favorable agreement with molecular dynamics simulational results and other high-resolution time resolved spectroscopic techniques. Evidence of the detection of the librational motion in methanol was not as convincing, however. While the solvent response function calculated for methanol showed features

that also appear in at approximately the same time in response functions produced by molecular dynamics simulations, their amplitudes were not larger than the sample deviation calculated from two replicate experiments.

Phase retrieval analysis was performed on single-angle broadband fluorescence upconversion data in an attempt to completely deconvolve the tempo-spectral characteristics of the gate pulse from the time resolved fluorescence signal. Because the phase retrieval algorithm assumes a that the input FROG traces adheres to the time-bandwidth produce relation that governs coherent pulses, it attempts to reproduce the bandwidth of the input FROG trace by modulating the amplitude of the trial electric field. Consequently, a new solvent response function had to be defined that attempted to minimized the effect of the amplitude modulation on the calculation of the instantaneous frequency. This new solvent response function was based on the time-windowed average frequency of the fluorescence signal. The solvent response function calculated for acetonitrile by this method showed some similarity to that calculated from the sum of Gaussians fitting method. When this method was applied to the methanol data, it produced less than satisfactory results. This as attributed to the significantly higher bandwidth fluorescence signal measured for C153 in methanol.

Lastly, in order to demonstrate the effect of performing a phase retrieval on non-coherently produced FROG traces several simulated FROG traces were produced. First, the accuracy of the phase retrieval algorithm/software implementation was demonstrated by simulating analytic pulses with a variety of temporal phase shapes. Next, FROG traces constructed to approximate the results of time resolved fluorescence Stokes shift experiments were constructed. Finally, a sum of analytically produced FROG traces

produces was used to simulate the emission from a simple distribution of solvated dye molecules. All of the FROG traces were input into the phase retrieval algorithm and the results of the retrieved electric fields were compared with the input fields.

For the FROG traces produced from analytic signals that obey the time-bandwidth product relation, the algorithm performed well producing fields that were nearly indistinguishable from the fields used to create the simulated FROG traces. The agreement of the retrieved fields with the input fields from a variety of different temporal phase shapes shows that the software implementation of the phase retrieval algorithm functions properly given the type of input fields for which it was designed.

The FROG traces that resembled time resolved fluorescence experiments proved to be more challenging to retrieve. The most striking feature of the retrieved electric fields from the non-coherently produced FROG traces was a substantial modulation of temporal amplitude. This modulation was also seen in the retrieved fields of the acetonitrile and methanol experimental results. It was concluded that the rate of modulation of the retrieved field is related to the bandwidth of the input FROG trace.

One consequence of the modulation in the retrieved amplitude is that it results in retrieved phases that have discontinuities. The discontinuities prevent the calculation of smooth time derivatives of the temporal phases. As introduced in the Experimental Results chapter, a time-windowed average was used to calculate the time derivative of the temporal phase. This time derivative was then related to the change in frequency of the input electric field, which was then shown to be similar to the fluorescence Stokes shift used to calculate the simulated FROG trace. The integrated Stokes shift of the input FROG trace was also mathematically related to the retrieved temporal phase. The

retrieved phase from FROG traces with a large Stokes shift and/or narrower bandwidth were shown to more closely resemble the input phases or Stokes shifts than FROG traces with smaller total Stokes shifts and/or higher bandwidths.

The transition from smooth retrieved electric fields to modulated fields was demonstrated. First, a simulated FROG trace was constructed from a Lorentzian lineshape function in the frequency domain. The Lorentzian lineshape was chosen because it resembles the emission from a single dye molecule subject to lifetime broadening. The retrieved phase from this simulated FROG trace was smooth and indistinguishable from the input phase. Next, two similar FROG traces were constructed from Lorentzian lineshapes with slightly different peak frequencies. Then, then two frequency shifted FROG traces were summed with the original FROG trace to simulate the incoherent addition of the emission of a distribution of several isolated dye molecules. The retrieved field of this summed, or composite, FROG trace was modulated in the same manner that the retrieved fields of the experimental data and simulated fluorescence Stokes shift FROG traces were. This further confirms that it is the violation of the time-bandwidth product relation of the input FROG traces that causes the algorithm to introduce modulation into the retrieved electric field.

Another finding was that the phase retrieval algorithm, as it was implemented for this work, does not converge to a unique solution when given a FROG trace that violates the time-bandwidth product relation. This was shown by initiating the retrieval of the same simulated FROG trace with three different seed pulses. The retrieved magnitudes were markedly different, However, Fourier analysis of the magnitudes showed that they were modulated similar at similar frequencies. Again, the frequency of the modulation is

likely related to the bandwidth of the input FROG trace. Although the modulation of the retrieved fields was not identical, the retrieved phases were similar to each other and to the phase of the input Lorentzian spectral fields.

In summary, broadband detection of the gated fluorescence from solvated Coumarin 153 is advantageous over traditional narrow bandwidth detection. The broadband techniques revealed oscillator structure of the fluorescence Stokes shift that has been predicted by molecular dynamics simulations, but not revealed with traditional narrow bandwidth techniques. Furthermore, broadband detection allows for the use of more sophisticated analysis, such as phase retrieval, of the time resolved fluorescence data. While application of phase retrieval analysis to the experimental data from this work was not completely successful, its application to physically relevant simulated FROG traces shows that this type of analysis holds some promise.

Bibliography

1. Fleming, G.R., *Chemical applications of ultrafast spectroscopy*. 1986, New York : Oxford [Oxfordshire] :: Oxford University Press ; Clarendon Press,.
2. Castner, E.W., M. Maroncelli, and G.R. Fleming, *Subpicosecond Resolution Studies of Solvation Dynamics in Polar Aprotic and Alcohol Solvents*. Journal of Chemical Physics, 1987. **86**(3): p. 1090-1097.
3. Damen, T.C. and J. Shah, *Femtosecond Luminescence Spectroscopy with 60-Fs Compressed Pulses*. Applied Physics Letters, 1988. **52**(16): p. 1291-1293.
4. Rosenthal, S.J., et al., *Femtosecond Solvation Dynamics in Acetonitrile - Observation of the Inertial Contribution to the Solvent Response*. Journal of Chemical Physics, 1991. **95**(6): p. 4715-4718.
5. Shah, J., *Ultrafast Luminescence Spectroscopy Using Sum Frequency Generation*. Ieee Journal of Quantum Electronics, 1988. **24**(2): p. 276-288.
6. Kahlow, M.A., et al., *Femtosecond Resolved Solvation Dynamics in Polar-Solvents*. Journal of Chemical Physics, 1989. **90**(1): p. 151-158.
7. Maroncelli, M. and G.R. Fleming, *Picosecond Solvation Dynamics of Coumarin-153 - the Importance of Molecular Aspects of Solvation*. Journal of Chemical Physics, 1987. **86**(11): p. 6221-6239.
8. Maroncelli, M., *The Dynamics of Solvation in Polar Liquids*. Journal of Molecular Liquids, 1993. **57**: p. 1-37.
9. Maroncelli, M., et al., *Solvent-Mediated Proton-Transfer Reactions*. Abstracts of Papers of the American Chemical Society, 1995. **210**: p. 99-PHYS.
10. Horng, M.L., et al., *Subpicosecond Measurements of Polar Solvation Dynamics - Coumarin-153 Revisited*. Journal of Physical Chemistry, 1995. **99**(48): p. 17311-17337.
11. Horng, M.L., J.A. Gardecki, and M. Maroncelli, *Rotational dynamics of coumarin 153: Time-dependent friction, dielectric friction, and other nonhydrodynamic effects*. Journal of Physical Chemistry A, 1997. **101**(6): p. 1030-1047.
12. Hazra, P., D. Chakrabarty, and N. Sarkar, *Solvation dynamics of Coumarin 153 in aqueous and non-aqueous reverse micelles*. Chemical Physics Letters, 2003. **371**(5-6): p. 553-562.
13. Luther, B.M., J.R. Kimmel, and N.E. Levinger, *Dynamics of polar solvation in acetonitrile-benzene binary mixtures: Role of dipolar and quadrupolar contributions to solvation*. Journal of Chemical Physics, 2002. **116**(8): p. 3370-3377.
14. Reynolds, L., et al., *Dipole solvation in nondipolar solvents: Experimental studies of reorganization energies and solvation dynamics*. Journal of Physical Chemistry, 1996. **100**(24): p. 10337-10354.

15. Lewis, J.E. and M. Maroncelli, *On the (uninteresting) dependence of the absorption and emission transition moments of coumarin 153 on solvent.* Chemical Physics Letters, 1998. **282**(2): p. 197-203.
16. Matyushov, D.V. and B.M. Ladanyi, *Nonlinear effects in dipole solvation .2. Optical spectra and electron transfer activation.* Journal of Chemical Physics, 1997. **107**(5): p. 1375-1387.
17. Ladanyi, B.M. and S. Klein, *Contributions of rotation and translation to polarizability anisotropy and solvation dynamics in acetonitrile.* Journal of Chemical Physics, 1996. **105**(4): p. 1552-1561.
18. Ladanyi, B.M. and M. Maroncelli, *Mechanisms of solvation dynamics of polyatomic solutes in polar and nondipolar solvents: A simulation study.* Journal of Chemical Physics, 1998. **109**(8): p. 3204-3221.
19. Stratt, R.M. and M. Maroncelli, *Nonreactive dynamics in solution: The emerging molecular view of solvation dynamics and vibrational relaxation.* Journal of Physical Chemistry, 1996. **100**(31): p. 12981-12996.
20. Bardeen, C.J., S.J. Rosenthal, and C.V. Shank, *Ultrafast solvation processes in polar liquids probed with large organic molecules.* Journal of Physical Chemistry A, 1999. **103**(49): p. 10506-10516.
21. Raineri, F.O., et al., *A Molecular Theory of Solvation Dynamics.* Journal of Chemical Physics, 1994. **100**(2): p. 1477-1491.
22. Ingrosso, F., et al., *Solvation dynamics in acetonitrile: A study incorporating solute electronic response and nuclear relaxation.* Journal of Physical Chemistry B, 2005. **109**(8): p. 3553-3564.
23. Rosenthal, S.J., et al., *Solvation Dynamics in Methanol - Experimental and Molecular- Dynamics Simulation Studies.* Journal of Molecular Liquids, 1994. **60**(1-3): p. 25-56.
24. Haacke, S., et al., *Improving the signal-to-noise ratio of femtosecond luminescence upconversion by multichannel detection.* Journal of the Optical Society of America B-Optical Physics, 1998. **15**(4): p. 1410-1417.
25. Diels, J.-C., *Ultrashort laser pulse phenomena : fundamentals, techniques, and applications on a femtosecond time scale / Jean-Claude Diels, Wolfgang Rudolph.*, ed. W. Rudolph, 1956-. 1996, San Diego :: Academic Press,.
26. Trebino, R. and D.J. Kane, *Using Phase Retrieval to Measure the Intensity and Phase of Ultrashort Pulses - Frequency-Resolved Optical Gating.* Journal of the Optical Society of America a-Optics Image Science and Vision, 1993. **10**(5): p. 1101-1111.
27. Trebino, R., et al., *Measuring ultrashort laser pulses in the time-frequency domain using frequency-resolved optical gating.* Review of Scientific Instruments, 1997. **68**(9): p. 3277-3295.
28. Kane, D.J. and R. Trebino, *Characterization of Arbitrary Femtosecond Pulses Using Frequency-Resolved Optical Gating.* Ieee Journal of Quantum Electronics, 1993. **29**(2): p. 571-579.
29. Trebino, R., *Frequency-resolved optical gating : the measurement of ultrashort laser pulses / Rick Trebino.*, in *Measurement of ultrashort laser pulses.* 2000, Kluwer Academic Publishers,: Boston :.

30. Langdon, B.T. and N.E. Levinger, *Recovery of time evolving fluorescence spectra via sum-frequency cross-correlation frequency resolved optical gating*. Applied Physics Letters, 2005. **87**(23).
31. RoperScientific, *Cosmic Rays and CCDs*. 2006.
32. Kovalenko, S.A., J. Ruthmann, and N.P. Ernsting, *Ultrafast Stokes shift and excited-state transient absorption of coumarin 153 in solution*. Chemical Physics Letters, 1997. **271**(1-3): p. 40-50.
33. Zhao, L.J., et al., *Femtosecond fluorescence spectroscopy by upconversion with tilted gate pulses*. Physical Chemistry Chemical Physics, 2005. **7**(8): p. 1716-1725.
34. Gustavsson, T., et al., *Femtosecond spectroscopic study of relaxation processes of three amino-substituted coumarin dyes in methanol and dimethyl sulfoxide*. Journal of Physical Chemistry A, 1998. **102**(23): p. 4229-4245.
35. Chapman, C.F., R.S. Fee, and M. Maroncelli, *Measurements of the Solute Dependence of Solvation Dynamics in 1-Propanol - the Role of Specific Hydrogen-Bonding Interactions*. Journal of Physical Chemistry, 1995. **99**(13): p. 4811-4819.
36. Cichos, F., R. Brown, and P.A. Bopp, *Coupled molecular dynamics/semiempirical simulation of organic solutes in polar liquids. II. Coumarin 153 in methanol and acetonitrile*. Journal of Chemical Physics, 2001. **114**(15): p. 6834-6842.
37. Ando, K., *Solvation dynamics and electronic structure development of coumarin 120 in methanol: A theoretical modeling study*. Journal of Chemical Physics, 1997. **107**(12): p. 4585-4596.
38. Ingrosso, F., B. Mennucci, and J. Tomasi, *Quantum mechanical calculations coupled with a dynamical continuum model for the description of dielectric relaxation: Time dependent Stokes shift of coumarin C153 in polar solvents*. Journal of Molecular Liquids, 2003. **108**(1-3): p. 21-46.
39. Matyushov, D.V., *Time-resolved fluorescence of polarizable chromophores*. Journal of Chemical Physics, 2001. **115**(19): p. 8933-8941.
40. Friedman, H.L., et al., *Applications of a Molecular Theory of Solvation Dynamics*. Journal of Physics-Condensed Matter, 1994. **6**: p. A131-A136.
41. Kumar, P.V. and M. Maroncelli, *Polar Solvation Dynamics of Polyatomic Solutes - Simulation Studies in Acetonitrile and Methanol*. Journal of Chemical Physics, 1995. **103**(8): p. 3038-3060.
42. Muhlpfordt, A., et al., *Coumarin 153 in the gas phase: optical spectra and quantum chemical calculations*. Physical Chemistry Chemical Physics, 1999. **1**(14): p. 3209-3218.
43. An, Q.R., W. Zinth, and P. Gilch, *In situ determination of fluorescence lifetimes via inverse Raman scattering*. Optics Communications, 2002. **202**(1-3): p. 209-216.
44. Ferrer, M.L., F. del Monte, and D. Levy, *Microviscosities at the porous cage of silica gel-glasses and ormosils through fluorescence anisotropy*. Journal of Physical Chemistry B, 2001. **105**(45): p. 11076-11080.

45. Ruthmann, J., et al., *Femtosecond relaxation of 2-amino-7-nitrofluorene in acetonitrile: Observation of the oscillatory contribution to the solvent response.* Journal of Chemical Physics, 1998. **109**(13): p. 5466-5468.
46. Prall, B.S., D.Y. Parkinson, and G.R. Fleming, *Probing correlated spectral motion: Two-color photon echo study of Nile blue.* Journal of Chemical Physics, 2005. **123**(5).
47. Baumann, R., et al., *Influence of confinement on the solvation and rotational dynamics of coumarin 153 in ethanol.* Journal of Physical Chemistry A, 2003. **107**(14): p. 2422-2430.

Appendix A

Sample output data file from the Labview data acquisition software

This is a printout of a sample output data file from the Labview data acquisition software described in Chapter 2. This sample shows the various text header and data fields, which output by the FROG data acquisition software. The descriptions of the fields and headers are denoted by <<description>>. The << and >> delimiters are not written by the software and are used here to offset the description from the information actually present in the data file.

The first line of the file is a text header that is supplied by the user in the Labview virtual instrument. It was designed to allow the user to enter data pertaining to the sample and instrument settings such as monochromator setting, SFG crystal angle and delay step size. The second line is written by the software and is the number of spectral points recorded by the CCD for each delay position. The next line is the number of CCD rows averaged to produce the output spectrum. Next is the integration time in milliseconds supplied to the CCD by the software, but ultimately set by the user at the beginning of the data acquisition run. Lastly, the delay position in femtoseconds is written as text with the spectrum following immediately. The spectrum is written in the binary 32 bit floating point number format with big-endian byte ordering. The next delay position is written

immediately after the 256th 32-bit floating point number spectral value, with its spectrum immediately following.

Sample Data File Printout

```
acn c153 200 nm 59.25 deg 9ps 20sec 4x4 <<user text header>>
256 <<number of points per spectrum>>
13 <<number of rows averaged to make one spectrum>>
20000ms <<text header indicating the integration time>>
50.04 <<delay in fs>>
<<binary data written to the file as 32 bit floating point number with
big-endian byte ordering>>
100.08
<<binary data>>
150.12
<<binary data>>
200.16
<<binary data>>
250.20
<<binary data>>
300.24
<<binary data>>
350.28
<<binary data>>
400.32
<<binary data>>


...


9007.20
<<binary data>>
```

Appendix B

Listing of Data Manipulation and Analysis Programs written in Igor Pro

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

Function MakeFirst(Ref)
      Wave/C Ref
      Duplicate/O/C Ref, firstEt
      firstEt=Sqrt(Exp(-((x/100)^2)))*Exp(-cmplx(0,enoise(1)))
End
Function makeSig(Ref,lastEt)
      Wave/C Ref, lastEt
      NVAR mDim, stT, dT
      Make/O/C/D/N=(mDim,mDim) EtSig
      SetScale/P x stT, dT,"", EtSig
      SetScale/P y stT, dT,"", EtSig
//    SetScale/P x 0,(dimdelta(ref,0)),"", EtSig
//    SetScale/P y 0,(dimdelta(ref,0)),"", EtSig
//    lastEt[0]=cmplx(0,0)
      EtSig=lastEt(x)*Ref(x-y)
//    rotate (mDim/2),Ref,lastEt
//    EtSig=lastEt[p]*Ref[p+q]
//    rotate (mDim/2),Ref,lastEt
//    SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig
//    SetScale/P y (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig


End

Function makeSig2(Ref,lastEt)
      Wave/C Ref, lastEt
      NVAR mDim, stT, dT
      Make/O/C/D/N=(mDim,mDim)  shuffle
      Matrixop /o EtSig=lastEt x Ref^t
      wave/c tmpsig=EtSig
      normWave(tmpsig)
      variable ii, rw=0
      duplicate/o/c lastEt, tmpRow
      //matrixop/o mag1=mag(EtSig)
      for (ii=0;ii<dimsize(tmpsig,0);ii+=1)
            tmpRow=tmpsig[ii][p]
            Rotate -ii, tmpRow
            tmpsig[ii][]=tmpRow[q]
```

```
	endfor
	//matrixop/o mag1=mag(EtSig)
	for (ii=mdim/2-1;ii>-1;ii-=1)//column manipulation
		shuffle[][rw]=tmpsig[p][ii]
		rw+=1
	endfor
	for (ii=mdim-1;ii>mdim/2-1;ii-=1)//column manipulation
		shuffle[][rw]=tmpsig[p][ii]
		rw+=1
	endfor
	tmpsig=shuffle
	SetScale/P x stT, dT,"", tmpsig
	SetScale/P y stT, dT,"", tmpsig
End

Function applyTrace(inWave,realTrace,i)
	Wave/C inWave //inWave is guessed efield matrix
	wave realTrace //matrix of square root of real data, same
dimensions as inWave
	variable i //iteration number
	wave tphase,tmag,gwave,g2wave //tphase isempty matrix to store
the phase

	FFT/COLS/DEST=tmp1 inWave //FFT each column
	wave tmp=tmp1
//	MatrixOP/o  tmp=r2polar(tmp) //transform to polar
	Matrixop/o tmag=mag(tmp)
//	matrixop tmag1=mag(inwave)
	//tmag=normWave(tmag)
	If (i==0)
		g2wave[0]=0.5
	endif
	variable g=OptimizeBenDeviation(tmag, realTrace, (g2wave[i-1]*2),
1e-5)
	//Matrixop/o tmag=tmag*g
	//gwave[i]=BenDeviation(tmag,realTrace,1)
	gwave[i]=BenDeviation(tmag,realTrace,g)
	If (i==0)
		gwave=BenDeviation(tmag,realTrace,g)
	endif
	g2wave[i]=g
	Matrixop/o tphase=phase(tmp1)
//	tphase=imag(tmp)
	MatrixOP/O tmp=cmplx(sqrt(realTrace),tphase)// replace mag of
guess with mag of real sig.
	MatrixOp/o tmp=p2rect(tmp)//transform back to rect
	IFFT/C/COLS /DEST=tmp2 tmp

	//print gwave[i]
End
Function applyTracePCGP(inWave,realTrace,i)
	Wave/C inWave //inWave is guessed efield matrix
	wave realTrace //matrix of square root of real data, same
dimensions as inWave
	variable i //iteration number
	wave tphase,tmag,gwave,g2wave //tphase isempty matrix to store
the phase
```

```
	FFT/COLS/DEST=tmp1 inWave //FFT each column
	wave/c tmp=tmp1
//	MatrixOP/o  tmp=r2polar(tmp) //transform to polar
	Matrixop/o tmag=mag(tmp)
//	matrixop tmag1=mag(inwave)
	//tmag=normWave(tmag)
	If (i==0)
		g2wave[0]=0.5
	endif
	variable g=OptimizeBenDeviation(tmag, realTrace, (g2wave[i-1]*2),
1e-5)
	//Matrixop/o tmag=tmag*g
	//gwave[i]=BenDeviation(tmag,realTrace,1)
	gwave[i]=BenDeviation(tmag,realTrace,g)

	If (i==0)
		gwave=BenDeviation(tmag,realTrace,g)
	endif

	g2wave[i]=g
	Matrixop/o tphase=phase(tmp)
//	tphase=imag(tmp)
	MatrixOP/O tmp2=p2rect(cmplx(sqrt(realTrace),tphase))// replace
mag of guess with mag of real sig.
	//MatrixOp/o tmp2=p2rect(tmp2)//transform back to rect
	IFFT/c/cols /DEST=EtSig tmp2

	//print gwave[i]
End

Function applySCtrace(inwave,realTrace,i, m)
	Wave/C inWave //inWave is guessed efield matrix
	wave realTrace //matrix of square root of real data, same
dimensions as inWave
	variable i,m //iteration number and avg sign
	wave tphase,tmag,gwave,g2wave //tphase isempty matrix to store
the phase

	FFT/COLS/DEST=tmp1 inWave //FFT each column
	wave tmp=tmp1
//	MatrixOP/o  tmp=r2polar(tmp) //transform to polar
	Matrixop/o tmag=mag(tmp)
	//tmag=normWave(tmag)
	If (i==0)
		g2wave[0]=0.5
	endif
	variable g=OptimizeBenDeviation(tmag, realTrace, (g2wave[i-1]*2),
1e-5)
	//Matrixop/o tmag=tmag*g
	//gwave[i]=BenDeviation(tmag,realTrace,1)
	gwave[i]=BenDeviation(tmag,realTrace,g)
	If (i==0)
		gwave=BenDeviation(tmag,realTrace,g)
	endif
	g2wave[i]=g
	Matrixop/o tphase=phase(tmp)
//	tphase=imag(tmp)
	if (m<1)
```

```
            Matrixop/o av1=tmp*(-3)
            MatrixOP/O tmp=cmplx(sqrt(realTrace),tphase)// replace mag
of guess with mag of real sig.
            MatrixOp/o tmp=p2rect(tmp)//transform back to rect
            Matrixop/o av1=av1-(3*tmp)
            IFFT/C/COLS /DEST=tmp2 tmp
      else
            wave aRef=av1
            Matrixop/o av1=av1+(tmp*4)
            MatrixOP/O tmp=cmplx(sqrt(realTrace),tphase)// replace mag
of guess with mag of real sig.
            MatrixOp/o tmp=p2rect(tmp)//transform back to rect
            Matrixop/o av1=(av1+(tmp*4))/2
            IFFT/C/COLS /DEST=tmp2 av1
//          Matrixop/o tmp=av

      endif


      //print gwave[i]

End

Function integrateRows(inWave)
      Wave/C inWave
      wave/C Eft, tmp2
      NVAR  stT, dT, mDim//, sigMaxRow
      variable i
      Eft=0
      for(i=0;i<mDim;i+=1)
            Eft=dT*inWave[p][i]+Eft[p]
      endfor
      //wavestats/Q/M=1 Eft

      //print V_maxloc
      //wavestats/Q tmp2
      //print V_maxRowLoc
      //rotate (x2pnt(tmp2,V_maxRowLoc)-x2pnt(Eft,V_maxloc)),Eft
      //rotate (sigmaxrow-x2pnt(Eft,V_maxloc)),Eft
      //rotate (mdim/2-x2pnt(Eft,V_maxloc)),Eft
      SetScale/P x stT,dT,"", Eft
End
Function masterCall(Et1,ref,realTrace,n, sc)
      wave/C Et1, ref   //Et1 is first calc. SFG eField, ref is gate
pulse
      wave realTrace    //realTrace is the real data matrix
      variable n, sc //number of iterations, use short cut GP 0=no
1=yest
      Variable timerRefNum1, timerRefNum2
      Variable microSeconds
      make/o/d/N=(n) gwave,g2wave
      timerRefNum1 = startMSTimer
      if (timerRefNum1 == -1)
            Abort "All timers are in use"
      endif
      variable i, m
      NVAR mDim
      make/O/N=(mdim) Eftmag, Eftphase
```

```
	duplicate/o/c Et1, currenteft
	//make/o/c /n=1 Etsig,Eft
	wave/C EtSig,Eft//,tmp2 //MatrixOP/O currentEft=Et1
	//initialize Eft so Et is not changed
	for (i=0;i<n; i+=1)
		makeSig(ref,currentEft) //makeSig(Ref,lastEt)
		normWave(EtSig)
		SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig
		SetScale/P y (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig

		if (sc==1)
			if((i+1)/3==trunc((i+1)/3))
				m=1
			Else
				m=0
			endif
			applySCtrace(Etsig,realTrace,i,m)
		else
			applyTrace(Etsig,realTrace,i)
		endif
		//EtSig=tmp2
		integrateRows(tmp2)
		currentEft=Eft//		integrateRows(EtSig)
		if (i/1==trunc(i/1))
			MatrixOP/o Eftmag=mag(Eft)//Eftmag=real(r2polar(Eft))
			MatrixOP/o
Eftphase=phase(Eft)//Eftphase=imag(r2polar(Eft))
			SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"",
Eftmag
			SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"",
Eftphase
			Unwrap 6.28319, Eftphase
			doUpdate                                // execute if
condition is TRUE
		endif
	endfor
	microSeconds = stopMSTimer(timerRefNum1)
	Print microSeconds/1e6, "seconds"
	Print "final error=", gwave[n]
//	Eftmag=real(r2polar(Eft))
//	Eftphase=imag(r2polar(Eft))
	//Unwrap 6.28319, Eftphase
End

Function masterPCGP(Et1,ref1,realTrace,n, sc)
	wave/C Et1, ref1//Et1 is first calc. SFG eField, ref1 is gate
pulse
	wave realTrace	//realTrace is the real data matrix
	variable n, sc //number of iterations, use short cut GP 0=no
1=yest
	Variable timerRefNum1, timerRefNum2
	Variable microSeconds, lowesterror=100
	make/o/d/N=(n) gwave,g2wave
	timerRefNum1 = startMSTimer
	if (timerRefNum1 == -1)
		Abort "All timers are in use"
	endif
```

```
      microSeconds = stopMSTimer(timerRefNum1) /////////
      variable i, m, ii
      NVAR mDim, sTt, dt
      make/O/N=(mdim) Eftmag, Eftphase
      wave/c Eft, besteft//wave EtSig,Eft, besteft//,tmp2
      //MatrixOP/O currentEft=Et1    //initialize Eft so Et is not
changed
      duplicate/o Et1, currenteft
      duplicate/o ref1, currentref
      wave /c neweft=currenteft
      make/c/o/d/n=(mdim) tmpRow
      make/d/o/c/n=(mdim,mdim) shuffle
      shuffle=cmplx(0,0)
      for (i=0;i<n; i+=1)
//          rotate mdim/2, currenteft
//          rotate mdim/2, currentref

            Matrixop/o EtSigNew=Normalize(currenteft x currentref^t)

            wave/c EtSig=EtSignew
            for (ii=0;ii<dimsize(EtSig,0);ii+=1)
                  tmpRow=EtSig[ii][p]

                  //print imag(Etsig[128][128])
                  Rotate -ii, tmpRow
                  EtSig[ii][]=tmpRow[q]
                  //if (imag(Etsig[128][128])==0)
                        //print ii
                  //Endif
            endfor
            //matrixop/o mag1=mag(EtSig)
            variable rw=0
            for (ii=mdim/2-1;ii>-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
            endfor
            for (ii=mdim-1;ii>mdim/2-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
            endfor
            EtSig=shuffle

//          SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig
//          SetScale/P y (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig

            applyTracePCGP(Etsig,realTrace,i)
//          wave/c tmp2
//          EtSig=tmp2


            rw=0
            for (ii=mdim/2-1;ii>-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
            endfor
            for (ii=mdim-1;ii>mdim/2-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
```

```
            endfor
            EtSig=shuffle

            //matrixop/o mag2=mag(EtSig)
            for (ii=0;ii<dimsize(EtSig,0);ii+=1)
                  tmpRow=EtSig[ii][p]
                  Rotate ii, tmpRow
                  EtSig[ii][]=tmpRow[q]
            endfor
            //matrixop/o mag2=mag(EtSig)
//          MatrixOp/o op=EtSig * EtSig^t
//          currenteft[]=op[p][0]*eft[p]
//          Matrixop/o currenteft=Normalize(currenteft)
//          MatrixOp/o currentEft=Normalize(Etsig x EtSig^t x (EtSig x
(EtSig^t x currentEft)))
            MatrixOp/o  currentEft=Normalize(EtSig x (EtSig^h x
currentEft))
            wave /c NewEft= currentEft
            wave /c Eft=currentEft
            SetScale/P x stT, dT,"", Eft
            If (sc==1)
                  MatrixOp/o currentref=Normalize(EtSig^t x EtSig x
EtSig^t x (EtSig x currentRef))
                  matrixOP/o ref=currentref
                  SetScale/P x stT, dT,"", ref
            Endif
//          integrateRows(tmp2)
//          currentEft=Eft//        integrateRows(EtSig)
            if (i/1==trunc(i/1))
                  MatrixOP/o
Eftmag=mag(NewEft)//Eftmag=real(r2polar(Eft))
                  MatrixOP/o
Eftphase=phase(NewEft)//Eftphase=imag(r2polar(Eft))
                  //Unwrap 6.28319, Eftphase
                  if (sc==1)
                        MatrixOP/o gatemag=mag(currentRef)
                        MatrixOP/o gatephase=phase(currentRef)
                  Endif
                  SetScale/P x 0,(dimdelta(ref1,0)),"", Eftmag
                  SetScale/P x 0,(dimdelta(ref1,0)),"", Eftphase
      //Unwrap 6.28319, Eftphase
                  //SetScale/P x 0,(dimdelta(ref,0)),"",
Eftmag//SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"", Eftmag
                  //SetScale/P x 0,(dimdelta(ref,0)),"",
Eftphase//SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"",
Eftphase                  //
                  Eftphase+=pi
                  Unwrap 6.28319, Eftphase
                  Eftphase-=pi
                  Eftphase[1,]=Eftphase[p]-Eftphase[0]
                  Eftphase[0]=0
                  FFT/OUT=4/DEST=Eft_FFT Eft

                  doUpdate                              // execute if
condition is TRUE
            endif
            If (gwave[i]<=lowesterror)
                  besteft=Neweft[p]
```

```
                  lowesterror=gwave[i]
            endif
      endfor
      MatrixOP/o gatephase=phase(currentRef)
      microSeconds = stopMSTimer(timerRefNum1)
      Print microSeconds/1e6, "seconds for PCGP fluor (1X application)"
      Print "lowest error=", lowesterror
      Print "final error=", gwave[n]

//    Eftmag=real(r2polar(Eft))
//    Eftphase=imag(r2polar(Eft))
      //Unwrap 6.28319, Eftphase
End
Function SVD(Et1,ref,realTrace,n, everyith)
      wave/C Et1, ref    //Et1 is first calc. SFG eField, ref is gate
pulse
      wave realTrace     //realTrace is the real data matrix
      variable n, everyith //number of iterations, use short cut GP
0=no 1=yest
      String filename,pathname
      Variable timerRefNum1, timerRefNum2
      Variable microSeconds, refnum, refnum2, refnum3
      make/o/d/N=(n) gwave,g2wave
      Newpath/Q /O data
      pathname = "data"
      Open/P=$pathname/T="IGTX"/Z=2 refnum as "EFields"
      fprintf refnum, "IGOR\rWAVES/D/C\tretrievals\rBEGIN\r"
      Open/P=$pathname/T="IGTX"/Z=2 refnum2 as "ewave"
      fprintf refnum2, "IGOR\rWAVES/D\titeration\tgwaveR\rBEGIN\r"
      Open/P=$pathname/T="IGTX"/Z=2 refnum3 as "FFTs"
      fprintf refnum3, "IGOR\rWAVES/D\tFFTR\rBEGIN\r"
      //wave /D gwave
      //Newpath/Q /O data
      //pathname = "data"
      //print (pathname)

      timerRefNum1 = startMSTimer
      if (timerRefNum1 == -1)
            Abort "All timers are in use"
      endif
      variable i, m, ii
      NVAR mDim, stT, dt
      make/O/N=(mdim) Eftmag, Eftphase
      wave EtSig,Eft//,tmp2    //MatrixOP/O currentEft=Et1
      //initialize Eft so Et is not changed
      duplicate/o/c Et1, currenteft
      duplicate/o/c ref, currentref, tmpRow
      make/o/d/c/n=(mdim,mdim) shuffle
      for (i=0;i<n; i+=1)
//          rotate mdim/2, currenteft
//          rotate mdim/2, currentref
            Matrixop /o EtSig=currenteft x currentref^t
            normWave(EtSig)
            matrixop/o mag1=mag(EtSig)
            for (ii=0;ii<dimsize(EtSig,0);ii+=1)
                  tmpRow=EtSig[ii][p]
                  Rotate -ii, tmpRow
                  EtSig[ii][]=tmpRow[q]
```

```
            endfor
            //matrixop/o mag1=mag(EtSig)
            variable rw=0
            for (ii=mdim/2-1;ii>-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
            endfor
            for (ii=mdim-1;ii>mdim/2-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
            endfor
            EtSig=shuffle

//          SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig
//          SetScale/P y (DimOffSet(ref,0)),(dimdelta(ref,0)),"", EtSig

            applyTracePCGP(Etsig,realTrace,i)
//          wave/c tmp2
//          EtSig=tmp2


            rw=0
            for (ii=mdim/2-1;ii>-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
            endfor
            for (ii=mdim-1;ii>mdim/2-1;ii-=1)//column manipulation
                  shuffle[][rw]=EtSig[p][ii]
                  rw+=1
            endfor
            EtSig=shuffle

            //matrixop/o mag2=mag(EtSig)
            for (ii=0;ii<dimsize(EtSig,0);ii+=1)
                  tmpRow=EtSig[ii][p]
                  Rotate ii, tmpRow
                  EtSig[ii][]=tmpRow[q]
            endfor
            matrixop/o mag2=mag(EtSig)
            //MatrixOp/o currentEft=Normalize((EtSig x EtSig^t) x
currentEft)

            matrixsvd /v=3 EtSig
            wave/c M_U
            currentEft=M_U[p][0]

            matrixop/o Eft=currentEft
            SetScale/P x stT, dT,"", Eft
//          MatrixOp/o currentref=Normalize((EtSig^t x EtSig) x
currentRef)
//          matrixOP/o ref=currentref
//          integrateRows(tmp2)
//          currentEft=Eft//        integrateRows(EtSig)
            if (i/everyith==trunc(i/everyith))
                  MatrixOP/o
Eftmag=mag(currentEft)//Eftmag=real(r2polar(Eft))
                  MatrixOP/o
Eftphase=phase(currentEft)//Eftphase=imag(r2polar(Eft))
```

```
                    //Unwrap 6.28319, Eftphase
//                  MatrixOP/o gatemag=mag(currentRef)
//                  MatrixOP/o gatephase=phase(currentRef)
                    SetScale/P x 0,(dimdelta(ref,0)),"",
Eftmag//SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"", Eftmag
                    SetScale/P x 0,(dimdelta(ref,0)),"",
Eftphase//SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"",
Eftphase              //
                    Eftphase+=pi
                    Unwrap 6.28319, Eftphase
                    Eftphase-=pi
                    Eftphase[1,]=Eftphase[p]-Eftphase[0]
                    Eftphase[0]=0
                    FFT/OUT=4/DEST=Eft_FFT Eft
                    //save/T/O/a=2 /p=$pathname Eft as "lasteft"
                    //wfprintf refNum, "\t%.12f\t%15.12f\r"  Eft
                    wfprintf refNum, "\t%.14g\t%.14g\r"  Eft

                    wfprintf refNum3, "\t%.14g\r"  Eft_FFT
                    doUpdate
              endif
              fprintf refNum2, "\t%g\t%.14g\r"  i, gwave[i]
      endfor
      MatrixOP/o gatephase=phase(currentRef)
      microSeconds = stopMSTimer(timerRefNum1)
      Print microSeconds/1e6, "seconds for PCGP-SVD"
      Print "final error=", gwave[n]
      fprintf refnum, "\rEND\r"
      fprintf refnum2, "\rEND\r"
      fprintf refnum3, "\rEND\r"
      Close refnum
      Close refnum2
      Close refnum3
//    Eftmag=real(r2polar(Eft))
//    Eftphase=imag(r2polar(Eft))
//    Unwrap 6.28319, Eftphase
End

function colGet(waveIn,ind)
      Wave waveIn
      Variable ind
      variable rows=dimsize (waveIn,0)//, cols=dimsize (waveIn,1)
      Make/O/D/N=(rows) E_Col
      E_Col=waveIn[p][ind]
End

Function BenDeviation(wave1, wave2, factor)
      Wave/c wave1, wave2
      Variable factor
      //wave BenDeviationTempWave
      //Duplicate/O wave1, BenDeviationTempWave
      MatrixOP/O BenDeviationTempWave =wave2-factor*powR(wave1,2)
//powR is faster than wave*wave
      MatrixOP/O BenDeviationTempWave =powR(BenDeviationTempWave,2)
      //BenDeviationTempWave =(wave2-factor*wave1^2)^2
      Variable result =
sqrt(matsum(BenDeviationTempWave))/numpnts(wave1)
```

```
	//KillWaves BenDeviationTempWave
	return result
end

Function BenDeviationForOptimize(w, x1)
	Wave w
	Variable x1

	SVAR wave1name
	SVAR wave2name
	Wave w1 = $wave1name
	wave w2 = $wave2name

	return BenDeviation(w1, w2, x1)
end

Function OptimizeBenDeviation(wave1, wave2, maxFactor, tolerance)
	Wave wave1, wave2
	Variable maxFactor
	Variable tolerance
	wave dummy
	//Make/N=1/O dummy
	String/G wave1name = GetWavesDataFolder(wave1, 2)
	String/G wave2name = GetWavesDataFolder(wave2, 2)
	Optimize/Q/L=0/H=(maxFactor)/T=(tolerance)
BenDeviationForOptimize, dummy
	if (V_flag)
		print "Error during optimize"+num2str(V_flag)
		return Nan
	endif

	//KillWaves dummy
	//KillStrings wave1name, wave2name

	return V_minloc
end

function mkGlobals(template)
	wave template//, data
	//Variable dim, startT, deltaT
	//wavestats/Q data
	Variable/G mDim=dimsize(template,0)
	Variable/G stT=dimoffset(template,0)
	Variable/G dT=dimdelta(template,0)//,
sigMaxRow=x2pnt(data,V_maxRowLoc)
	make/o/d/n=(mDim,mDim) tPhase, tMag
	duplicate/o template, Eft, BenDeviationTempWave, bestEft
	redimension /R BenDeviationTempWave
	Make/N=1/O dummy
End

function mkFake(fake)
	wave/c fake
//	FFT/ROWS/DEST=fakeTrace fake
	FFT/COLS/DEST=fakeTrace fake
	fakeTrace=fakeTrace^2 //+enoise(2)+2
End
```

```
macro fkspec()

	//fluor=cmplx(exp(-(x/30)^2),x/50)
	makeSig(ef256,fluor)
	mkFake(EtSig)
	fakephase=imag(r2polar(fluor))

	//Unwrap 6.28319, fakephase
	fakemag=real(r2polar(fluor))

End


function normWave(inWave)
	wave inWave
	wavestats/Q/M=1 inWave
	matrixOP/O inWave=inWave/V_max
End
constant kCSpeed=299792458e9 //c in nm/s

// Assumes that inWave has columns of spectra, each column taken at a
different time.
// The wave scaling is in units which match the wavelength to the
constant kCSpeed above.

Function convertSpectra2(inWave,N)// still doesn't pass row scaling
	Wave inWave
	Variable N
	Variable Lstart=DimOffset(inWave,0)
	//Variable oldN=DimSize(inWave,0)
	Variable Lend=LStart+DimSize(inWave,0)*DimDelta(inWave,0)
	Variable deltaNu=1078389655502.22/kCSpeed
	// Create a wave for the new frequencies.  Assuming
Fstart=kCSpeed/Lstart...
	// this gives the "old scaling" using the wavelength limits of
inwave
	// Make/O/N=(oldN) wavelengths=1/(1/Lend+p/oldN/Lstart-
p/oldN/Lend)
	Variable oldN=floor(1/deltaNu*(1/Lstart-1/Lend))
	Make/O/N=(oldN) wavelengths=1/(1/Lend+p*deltaNu)
	Variable cols=DimSize(inWave,1)

	// Create the output wave
	//Make/O/N=(oldN,cols) outWave
	Duplicate/O inWave, outWave
	Redimension/N=(oldN,-1) outWave
	Make/O/N=(oldN) tmp
	Make/O/N=(Dimsize(inWave,0)) xwave

	xwave=LStart+p*DimDelta(inWave,0)
	Variable i
	for(i=0;i<cols;i+=1)
		ImageTransform/G=(i) GetCol inWave
		Wave W_ExtractedCol
		tmp=interp((wavelengths[p]),xwave, W_ExtractedCol)
		ImageTransform/G=(i)/D=tmp putCol outWave
	endfor
	insertpoints/M=0 0,Round((N-oldN)/2), outWave
```

```
      insertpoints/M=0 (DimSize(outWave,0)),(N-
DimSize(outWave,0)),outWave
      SetScale/P x  ((1/wavelengths[0]-Round((N-
oldN)/2)*deltaNu)*kcspeed),(deltaNu*kcspeed),"", outWave
//(kCSpeed/Lend),(kCSpeed/LStart)
End


Function zapNaN(inwave)
      wave inWave
      variable i, ii
      for(i=0;i<dimsize(inWave,0);i+=1)
            for(ii=0;ii<dimsize(inWave,1);ii+=1)
                  if (inWave[i][ii]<0)
                        inWave[i][ii]=0
                  endif
            endfor
      endfor

End

Function Matsum(inwave)
      wave inWave
      variable i, ii
      variable sm=0

      for(i=0;i<dimsize(inWave,0);i+=1)
            for(ii=0;ii<dimsize(inWave,1);ii+=1)
                  sm+=inwave[i][ii]
            endfor
      endfor
      return sm
End
Macro GetDimensions() : Panel
      PauseUpdate; Silent 1          // building window...
      NewPanel/K=0 /W=(279,90,610,320) as "Get Dimensions"
      SetVariable
Dim_setDt,pos={36,27},size={137,18},title="dt/fs",fSize=12,format="%g"
      SetVariable Dim_setDt,value= dT,bodyWidth= 100
      SetVariable Dim_setStT,pos=[43],size={166,18},title="start
t/fs",fSize=12
      SetVariable Dim_setStT,format="%g",value= stT,bodyWidth= 100
      SetVariable Dim_setmDim,pos=[44],size=[42],title="Grid
Dim",fSize=12
      SetVariable Dim_setmDim,format="%g",value= mDim,bodyWidth= 100
      CheckBox Dim_setcheckTmplt,pos=[45],size=[46],title="Use wave as
template"
      CheckBox Dim_setcheckTmplt,value= 0, variable=usetmplt
      PopupMenu Dim_setpopuptmplt,pos={84,131},size=[43]
      PopupMenu Dim_setpopuptmplt,mode=1,popvalue="gate",value=
#"WaveList(\"*\",\";\",\"\") "
      Button
Dim_setbuttontmplt,pos={85,160},size=[37],proc=Dim_setgetDims,title="Ge
t 'em"
EndMacro
Function Dim_setgetDims(ctrlName) : ButtonControl
      String ctrlName
      String gwave
```

```
	NVAR usetmplt
	if(usetmplt==1)
		Controlinfo Dim_setpopuptmplt
		gwave=S_value
		mkGlobals($gwave)
	endif
End

Function FreqScaling(mono,nop)
	Variable mono,nop
	variable ind
	mono=mono-30.24
	Make/O /N=(nop) /D FqWave
	for(ind=0;ind<nop;ind+=1)
		FqWave[ind]=2.99792458E8/(mono+(ind*0.24))/1E-9
	endfor
End
Macro XFROG() : Panel
	PauseUpdate; Silent 1		// building window...
	variable/g dT, stT, mDim, stL=0,endL=0, dL=0.512673, monoSet,
timeWin, twinErr=0,oldTS=16.68,catZero=0,usetmplt, prev=0, meth
	variable/g storeProg, everyith, maxh, blueL,monoset=200
	string/g header1=" ", header2=" "
	NewPanel/K=1 /W=(410,44,810,497) as "X-FROG Tools"
	ModifyPanel frameStyle=3, frameInset=3
	TabControl resample,pos=[37],size={384,436},proc=TabProc1
	TabControl resample,tabLabel(0)="Resample Trace/Gate"
	TabControl resample,tabLabel(1)="Make Unix Files",value= 0
	CheckBox
Mkfile_prev,pos={81,126},size={261,17},disable=1,title="Use results
from previous retrieval"
	CheckBox Mkfile_prev,help={"This will use the out.txt file from
the last retrieval run"}
	CheckBox Mkfile_prev,fSize=14,variable= prev
	SetVariable
Res_dL,pos={175,140},size={162,20},title="nm/pixel",fSize=14
	SetVariable Res_dL,format="%g",value= dL,bodyWidth= 100
	TitleBox Res_limitLable,pos={123,223},size={243,17},title="place
cursors for wavelength limits"
	TitleBox Res_limitLable,fSize=14,frame=0
	Button
Res_csr,pos={228,247},size={80,20},proc=Res_goCsr,title="Get Limits"
	Button Res_csr,fSize=14
	Button Res_csr,help={"The FROG trace that you wish to sample must
be the topmost plot window"}
	Button
Res_button,pos={237,378},size={80,20},proc=Res_goResam,title="Resample"
	Button Res_button,fSize=14
	SetVariable Res_dim,pos={95,326},size=[47],title="New Grid
Dimension"
	SetVariable Res_dim,help={"must be a power of
2"},fSize=14,format="%d"
	SetVariable Res_dim,limits={128,inf,128},value= mDim,bodyWidth=
100
	PopupMenu Res_popuptmplt,pos={143,34},size={158,24},title="FROG
Trace",fSize=16
	PopupMenu Res_popuptmplt,mode=4,popvalue="trace1",value=
#"WaveList(\"*\",\";\",\"DIMS:2\") "
```

```
	ValDisplay Res_stL,pos={179,277},size={75,14},title="start"
	ValDisplay Res_stL,limits={0,0,0},barmisc={0,1000},value= #"stL"
	ValDisplay Res_endL,pos={264,277},size={65,14},title="end"
	ValDisplay Res_endL,limits={0,0,0},barmisc={0,1000},value=
#"endL"
	SetVariable
Res_tmStp,pos={53,87},size={284,20},title="experimental time step/fs"
	SetVariable Res_tmStp,fSize=14,format="%g"
	SetVariable Res_tmStp,limits={16.68,inf,16.68},value=
oldTS,bodyWidth= 100
	CheckBox Res_Zero,pos={218,352},size={116,14},title="Center Time
axis at 0"
	CheckBox Res_Zero,variable= catZero
	PopupMenu Res_gatepopup,pos={153,59},size={140,24},title="Gate
Pulse",fSize=16
	PopupMenu Res_gatepopup,mode=1,popvalue="back",value=
#"WaveList(\"*\",\";\",\"DIMS:1\") "
	PopupMenu
Mkfile_meth,pos={89,157},size={198,21},disable=1,title="Retrieval
Algorithm"
	PopupMenu Mkfile_meth,fSize=14
	PopupMenu Mkfile_meth,mode=1,popvalue="Vanilla",value=
#"\"Vanilla;PCGP\""
	SetVariable
Mkfile_iter,pos={80,191},size={217,20},disable=1,title="Number of
iterations"
	SetVariable Mkfile_iter,fSize=14,format="%d",value=
maxh,bodyWidth= 70
	Button
Mkfile_button,pos={148,371},size={100,20},disable=1,proc=mkfile_go,titl
e="Make Data File"
	CheckBox
Mkfile_storeProg,pos={79,245},size={276,17},disable=1,title="Store
fluorescence fields periodically"
	CheckBox Mkfile_storeProg,help={"This will save the progress of
the retrieval in the file prog.txt"}
	CheckBox Mkfile_storeProg,fSize=14,variable= storeProg
	SetVariable
Mkfile_everyith,pos={82,277},size={157,20},disable=1,title="Store
every"
	SetVariable Mkfile_everyith,fSize=14,format="%d",value=
everyith,bodyWidth= 70
	SetVariable Res_monoset,pos={98,113},size={199,20},title="Mono.
setting in nm"
	SetVariable Res_monoset,fSize=14,value= monoset,bodyWidth= 60
	TitleBox
Mkfile_ith,pos={250,280},size={80,17},disable=1,title="ith iteration"
	TitleBox Mkfile_ith,fSize=14,frame=0
	Button
Res_Setscale,pos={212,166},size={80,20},proc=Res_goReSc,title="Set
Scale"
	Button Res_Setscale,help={"Sets the scale of the input trace to
nm and fs"}
	Button Res_Setscale,fSize=14
	GroupBox Res_1,pos={16,31},size={367,169}
	GroupBox Res_2,pos={16,210},size={367,95}
	GroupBox Res_3,pos={16,315},size={367,95}
```

```
	SetVariable
Mkfile_header1,pos={17,49},size={354,15},disable=1,title="header info"
	SetVariable Mkfile_header1,help={"Enter any text you want (no
returns and no curly brackets {})"}
	SetVariable Mkfile_header1,limits={-inf,inf,0},value=
header1,bodyWidth= 300
	SetVariable
Mkfile_header2,pos={17,78},size={354,15},disable=1,title="header info"
	SetVariable Mkfile_header2,help={"Enter any text you want (no
returns and no curly brackets {})"}
	SetVariable Mkfile_header2,limits={-inf,inf,0},value=
header2,bodyWidth= 300
	GroupBox Mkfile_1,pos={13,36},size={372,71},disable=1
	GroupBox Mkfile_2,pos={13,121},size={372,103},disable=1
	GroupBox Mkfile_3,pos={13,237},size={372,71},disable=1
EndMacro

Function TabProc1(ctrlName,tabNum) : TabControl
	String ctrlName
	Variable tabNum
	// note: disable=0 means "show", disable=1 means "hide"
	If (tabNum==2)//resample Gate
		ModifyControlList ControlNameList("",";","Mkfile_*")
disable=1
		ModifyControlList ControlNameList("",";","Res_*") disable=1
		ModifyControlList ControlNameList("",";","ResG_*")
disable=0
	Endif

	If (tabNum==0)//resample trace
		ModifyControlList ControlNameList("",";","Mkfile_*")
disable=1
		ModifyControlList ControlNameList("",";","Res_*") disable=0
		ModifyControlList ControlNameList("",";","ResG_*")
disable=1
	Endif
	If (tabNum==1)//make files for unix
		ModifyControlList ControlNameList("",";","Mkfile_*")
disable=0
		ModifyControlList ControlNameList("",";","Res_*") disable=1
		ModifyControlList ControlNameList("",";","ResG_*")
disable=1
		//DrawPict 0,0,1,1,ProcGlobal#dude
	Endif
	return 0
End
Function Res_goResam(ctrlName) : ButtonControl
	String ctrlName
	String resamName, resamGate
	NVAR stL, mDim, dl, monoset,timeWin, endL
	//stL=monoset
	Controlinfo Res_popuptmplt
	resamName=S_value
	Controlinfo Res_gatepopup
	resamGate=S_value

	//print "time window=",timeWin
	ReSam1($resamName,$resamGate,mDim,stL,endL,dL,timeWin)
```

```
End

Function Res_goReSc(ctrlName) : ButtonControl
	String ctrlName
	string resamName
	NVAR monoset, blueL, dl, oldTS
	Controlinfo Res_popuptmplt
	resamName=S_value
	blueL=monoset*1.9986 - 126.5
	SetScale /P x blueL, dl,"", $resamName
	SetScale /P y 0, oldTS,"", $resamName
End

Function Res_goCsr(ctrlName) : ButtonControl
	String ctrlName
	NVAR stL, endL



	if (xcsr(a)<xcsr(b))
		stL=xcsr(a)
		endL=xcsr(b)
	else
		stL=xcsr(b)
		endL=xcsr(a)
	endif
//	print "start=", stL
//	print "end=", endL
end

End
Function ReSam1(inWave,inwaveGate,N,Lstart,Lend,deltalam,twin)// still
doesn't pass row scaling
	Wave inWave
	wave/c inwaveGate
	Variable N, Lstart,Lend, deltaLam,tWin
	//Lstart *=2 // used for 600 groove per mm grating
	//Lstart-=126.77//30.24
	NVAR twinErr
	//Variable Lend=LStart+DimSize(inWave,0)*deltaLam
	Variable Nu=kcspeed*(1/Lstart-1/Lend)
	//Variable deltaNu=1078389655502.22/kCSpeed
	// Create a wave for the new frequencies.  Assuming
Fstart=kCSpeed/Lstart...
	// this gives the "old scaling" using the wavelength limits of
inwave
	// Make/O/N=(oldN) wavelengths=1/(1/Lend+p/oldN/Lstart-
p/oldN/Lend)
	Variable oldN=abs(pcsr(a)-pcsr(b))
	Make/d/O/N=(N) wavelengths=1/(1/Lend+p*(1/Lstart-1/Lend)/N)
	if (oldN<=N)//Dimsize(inwave,0))
		twinErr=0
		Variable cols=DimSize(inWave,1)
		// Create the output wave
		Make/O/d/N=(N,cols) outWave1
		//Duplicate/O inWave, outWave
		//Redimension/N=(oldN,-1) outWave
		Make/d/O/N=(N) tmp
```

```
            Make /d/O/N=(Dimsize(inwave,0)) xwave

            xwave=Dimoffset(inwave,0)+p*deltaLam
            Variable i
            for(i=0;i<cols;i+=1)
                  ImageTransform/G=(i) GetCol inWave
                  Wave W_ExtractedCol
                  //Interpolate2/T=1/N=1024/I=3/Y=tmp/X=wavelengths
xwave, W_ExtractedCol
                  tmp=interp((wavelengths[p]),xwave, W_ExtractedCol)
                  ImageTransform/G=(i)/D=tmp putCol outWave1
            endfor

//          insertpoints/M=0 0,Round((N-oldN)/2), outWave
//          insertpoints/M=0 (DimSize(outWave,0)),(N-
DimSize(outWave,0)),outWave
            NVAR oldTS
            Make/d/O/N=(Dimsize(inWave,1)) ywave
            Make/d/O/N=(N) tmp
            SetScale /I x 0, 1e15*N/(nu),"", tmp       //tmp is now
scaled in fs not sec
            ywave=p*oldTS
            Make/o/d/n=(N,N) outwave2
            for(i=0;i<N;i+=1)
                  ImageTransform/G=(i) GetRow outwave1
                  Wave W_ExtractedRow
                  //tmp=interp(x,ywave, W_ExtractedRow)
                  Interpolate2/T=2/I=3/Y=tmp ywave, W_ExtractedRow
                  ImageTransform/G=(i)/D=tmp putRow outWave2
            endfor

            NVAR catZero
            duplicate/O outwave2 outwave
            //SetScale/P x ((kCSpeed/Lend)-Round((N-oldN)/2)*Nu/N)
,Nu/N,"", outWave   //(kCSpeed/Lend),(kCSpeed/LStart)
            SetScale/I x kcspeed/Lend,kcspeed/Lstart,"", outwave
            wavestats/q outwave
            outwave/=V_max
            If (catZero==1)
                  SetScale/P y  (-1e15*N/(nu)/2),(1e15/(nu)),"",
outWave
            else
                  SetScale/P y  0,(1e15/(nu)),"", outWave
            endif

            zapNan(outwave)
            NVAR stT,dT
            stT=-1e15*N/(nu)/2
            dT=1e15/(nu)
            Make/o/c/d/n=(N) outRef
            SetScale/P x  (stT),(dT),"", outRef
            variable stind=floor((Dimoffset(inwaveGate,0)-stT)/dT)
            If (Dimoffset(inwaveGate,0)>stT)
                  //outRef[0,(stind-1)]=cmplx(0,0)
                  outRef=cmplx(0,0)

      outRef[(stind),(x2pnt(outRef,rightx(inwaveGate)))]=inwaveGate(pnt
2x(outRef,p))
```

```
                  //outRef[(x2pnt(rightx(inwaveGate))),(N-
1)]=cmplx(0,0)
                  print stind
                  print x2pnt(outRef,rightx(inwaveGate))
            else

            Endif

      else
            twinErr=1
            Print "Error: Reduce Time Window"
      Endif
      killwaves tmp,xwave,ywave, w_ExtractedCol,
w_extractedRow,outwave2, outwave1//, wavelengths
End

Function mkfile_go(ctrlName): ButtonControl
      String  ctrlName
      String flName
      variable f1
      NVAR mDim, meth, storeProg, everyith, maxh, prev
      SVAR header1, header2
      open f1 as "xfrog_pfile.txt"
      If(stringmatch(S_fileName,""))
            print ""
            print "no file was written"
            print ""
      else
            print header1+ " " + header2
            //header1="<<header>>" + header1 +" "
            //fprintf f1, "{" + header1
            //fprintf f1, header2
            //fprintf f1, "}\r"
            fprintf f1,  "%d\r", mDim //write dimension
            fprintf f1, "%d\r", prev //write use previous retrieval
            controlinfo Mkfile_meth
            If (stringmatch(S_value,"Vanilla")==1)
                  meth=0
            else
                  meth=1
            Endif
            fprintf f1,"%d\r", meth //write method
            fprintf f1,"%d\r", maxh //write number of iterations
            fprintf f1,"%d\r", storeProg //write store progress 1=yes
0=no
            fprintf f1,"%d\r", everyith    //write how often to store
progress

            //reverse /DIM=0/p outwave
            rotate (mdim/2), outwave
            wfprintf f1, "" outwave// write resampled FROG trace
            wfprintf f1, "" outRef//write resampled gate field
            close f1

            rotate -(mdim/2), outwave
            //reverse /DIM=0/p outwave
            If (prev==1)
```

```
                  print "Using results from a previous retrieval stored
in out.txt"
            endif
            print "Retrieval method is", S_Value
            print "Retrieval will run for", maxh, "iterations"
            If (storeProg==1)
                  print "Storing fluorescence fields after every",
everyith, "iteration in prog.txt"
            Endif

      endif
end



////////CCD data loader part
#include <Concatenate Waves>
Menu "XFROG"
      "Load XFROG data", AutoLoadData()
      "Open XFROG Panel", XFROG()
End

//
//
//
//
//
//
Function AutoLoadData()
      String pathName="_New Path_", matPathName="_New Path_"
      Variable refNum, xDim, yDim, n=0, bCount
      String sample, exInfo, tim, matBase, matName, xd, yd
      String fileName
      Variable fileIndex=0, gotFile,
fileNameLength,headLength,V_filePos
      String xName, yName
      Variable timv

      Prompt pathName, "Name of path containing text files", popup
PathList("*", ";", "")+"_New Path_"
      DoPrompt "Load CCD", pathName
      if (V_flag != 0)
            return -1
      // User cancelled.
      endif

      if (CmpStr(pathName, "_New Path_") == 0)         // User selected
new path ?
            NewPath/O data
      // This brings up dialog and creates or overwrites path
            if (V_flag != 0)
                  return -1
      // User cancelled
            endif
            pathName = "data"
      endif
```

```
//    Prompt matPathName,"Time-Wavelength Path (must be different from
original data)" , popup PathList("*", ";", "")+"_New Path_"
//    DoPrompt "Path to store resulting Time-Wavelenth Matrix",
matPathName
//    if (V_flag != 0)
//          return -1
      // User cancelled.
//    endif

//    if (CmpStr(matPathName, "_New Path_") == 0)            // User
selected new path ?
//          NewPath/O matData
      // This brings up dialog and creates or overwrites path
//          if (V_flag != 0)
//                return -1
      // User cancelled
//          endif
//          matPathName = "matData"
//    endif

                  Variable err = V_flag



      do
            fileName = IndexedFile($pathName, fileIndex, "????")
      // Get name of next text file in path
            print fileName
            gotFile = CmpStr(fileName, "")

            if (gotFile)

                  Open/R/Z=2/P=$pathName refNum as fileName

                  //print S_fileName

                  FReadLine refNum, sample
                  FReadLine refNum, xd
                  FReadLine refNum, yd
                  FReadLine refNum, exInfo
                  print xd
                  if (fileIndex==1)
                        xDim=str2num(xd)
                        yDim=1
                        Prompt xDim, "Number of Serial Points"
                        DoPrompt "Enter X Number of Serial Points",
xDim
                        if (V_flag != 0)
                              return -1
            // User cancelled
                  endif

                  Prompt matBase,"Base Name"
                  DoPrompt "Enter Time-Wavelength Matrix Base
Name", matBase

                  if (V_flag != 0)
```

```
                        return -1
            // User cancelled
                    endif

                endif
                Print xDim
                bCount=xDim*yDim*2
                headLength=V_filePos
                Variable ind=0,temp,imInd=0
                Make /O/N=0 /D tempMat // D (double presision)
                do
                    FReadLine refNum, tim
                    timv=str2num(tim)
                    tim=num2str(timv)
                    Printf "Time step = %s\r", tim
                    FStatus refNum
                    n=V_filePos
                    FSetPos refnum, n
                    Make /O/N=(bCount/2) /D image // D (double
presision) used to be W (Integer)
                    ind=0
                    imInd=imInd+1

                    do
                        FBinRead /F=4 refNum, temp     // try /B=1
or 3 ****************************
                        image[ind]=temp
                        ind=ind+1
                    while (ind<=bCount/2 -1)
      // as long as expression is true

                    Redimension/N=(xDim,0) image //substituded 0
for ydim

                    ConcatenateWaves("tempMat","image") //Duplicate
image, $tim
                    Killwaves image
                    FStatus refnum
                while (V_filePos< V_logEOF)                    // as
long as expression is true
                Close refNum
                matName=matBase+num2istr(fileIndex-1)
                print matName
                Redimension/N=(xDim,imInd) tempMat
                Duplicate tempMat,$matName
                //Save/C/P=matPathName tempMat as matName
                Killwaves tempMat
                fileIndex += 1
            endif


      while (gotFile)
      // Until IndexedFile runs out of files
End
Function LR()
      String pathName="_New Path_", matPathName="_New Path_"
      Variable refNum, xDim, yDim, n=0, bCount
      String sample, exInfo, tim, matBase, matName, xd, yd
```

```
	String fileName
	Variable fileIndex=0, gotFile,
fileNameLength,headLength,V_filePos
	String xName, yName
	Variable timv

	Prompt pathName, "Name of path containing text files", popup
PathList("*", ";", "")+"_New Path_"
	DoPrompt "Load CCD", pathName
	if (V_flag != 0)
		return -1
	// User cancelled.
	endif

	if (CmpStr(pathName, "_New Path_") == 0)		// User selected
new path ?
		NewPath/O data
	// This brings up dialog and creates or overwrites path
		if (V_flag != 0)
			return -1
	// User cancelled
		endif
		pathName = "data"
	endif
	//print pathname
	LoadWave /G/D/A=Gerror /P=$pathName "z.txt"
	LoadWave /G/D/B="C=1,F=0,T=4,N=ret;
C=1,F=0,T=4,N=imt;"/P=$pathName "out.txt"

End

Function SVD2(Et1,ref,realTrace,n, sc)
	wave/C Et1, ref	//Et1 is first calc. SFG eField, ref is gate
pulse
	wave realTrace	//realTrace is the real data matrix
	variable n, sc //number of iterations, use short cut GP 0=no
1=yest
	Variable timerRefNum1, timerRefNum2
	Variable microSeconds
	make/o/d/N=(n) gwave,g2wave
	timerRefNum1 = startMSTimer
	if (timerRefNum1 == -1)
		Abort "All timers are in use"
	endif
	variable i, m, ii
	NVAR mDim, stT, dt
	make/O/N=(mdim) Eftmag, Eftphase
	wave EtSig,Eft//,tmp2	//MatrixOP/O currentEft=Et1
	//initialize Eft so Et is not changed
	duplicate/o Et1, currenteft
	duplicate/o ref, currentref, tmpRow
	make/o/c/n=(mdim,mdim) shuffle
	for (i=0;i<n; i+=1)
		Matrixop /o EtSig=currenteft x currentref^t
		normWave(EtSig)
		matrixop/o mag1=mag(EtSig)
		for (ii=0;ii<dimsize(EtSig,0);ii+=1)
			tmpRow=EtSig[ii][p]
```

```
                Rotate -ii, tmpRow
                EtSig[ii][]=tmpRow[q]
        endfor
        variable rw=0
        for (ii=mdim/2-1;ii>-1;ii-=1)//column manipulation
                shuffle[][rw]=EtSig[p][ii]
                rw+=1
        endfor
        for (ii=mdim-1;ii>mdim/2-1;ii-=1)//column manipulation
                shuffle[][rw]=EtSig[p][ii]
                rw+=1
        endfor
        EtSig=shuffle

        applyTracePCGP(Etsig,realTrace,i)

        rw=0
        for (ii=mdim/2-1;ii>-1;ii-=1)//column manipulation
                shuffle[][rw]=EtSig[p][ii]
                rw+=1
        endfor
        for (ii=mdim-1;ii>mdim/2-1;ii-=1)//column manipulation
                shuffle[][rw]=EtSig[p][ii]
                rw+=1
        endfor
        EtSig=shuffle
        for (ii=0;ii<dimsize(EtSig,0);ii+=1)
                tmpRow=EtSig[ii][p]
                Rotate ii, tmpRow
                EtSig[ii][]=tmpRow[q]
        endfor
        matrixop/o mag2=mag(EtSig)

        matrixsvd  EtSig
        wave/c M_U, M_VT
        currentEft=M_U[p][0]
        currentref=M_VT[0][p]
        matrixop/o Eft=currentEft
        SetScale/P x stT, dT,"", Eft
        if (i/1==trunc(i/1))
                MatrixOP/o
Eftmag=mag(currentEft)//Eftmag=real(r2polar(Eft))
                MatrixOP/o
Eftphase=phase(currentEft)//Eftphase=imag(r2polar(Eft))
                SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"",
Eftmag
                SetScale/P x (DimOffSet(ref,0)),(dimdelta(ref,0)),"",
Eftphase              //
                Eftphase+=pi
                Unwrap 2*pi, Eftphase
                Eftphase-=pi
                MatrixOP/o gatephase=phase(currentRef)
                MatrixOP/o gatemag=mag(currentRef)
                gatephase+=pi
                unwrap 2*pi, gatephase
                gatephase-=pi
                doUpdate                                // execute if
condition is TRUE
```

```
            endif
        endfor


        microSeconds = stopMSTimer(timerRefNum1)
        Print microSeconds/1e6, "seconds for PCGP-SVD"
        Print "final error=", gwave[n]
End

function spectro(win, inwave,sq)
        Variable win, sq
        Wave inwave
        wavestats/q inwave
        make/o/d/n=(V_npnts-win,V_npnts) spectro_M
         sq+=3
        variable n
        for (n=0;n<(V_npnts-win);n+=1)
              FFT/OUT=(sq)/PAD={V_npnts}/RP=[n,n+win]/DEST=tmpFFT inwave
              spectro_M[n][]=tmpFFT[q]
        endfor
        SetScale/P x 0,dimdelta(inwave,0),"", spectro_M
        SetScale/P y dimoffset(tmpFFT,0),dimdelta(tmpFFT,0),"", spectro_M
        string newname= nameofwave(inwave)+"_spec"
        duplicate/o spectro_M, $newname
        killwaves tmpFFT, spectro_M
End

function pm(inwave)
        Wave/c inwave
        wavestats/q inwave
        duplicate/o inwave, Ptmp
        redimension /r Ptmp
        Ptmp=atan2(imag(inwave),real(inwave))
        Ptmp+=pi
        unwrap 2*pi, Ptmp
        Ptmp-=pi
        string newname= nameofwave(inwave)+"Phs"
        duplicate/o Ptmp, $newname
        Ptmp=cabs(inwave)
        newname= nameofwave(inwave)+"Mag"
        duplicate/o Ptmp, $newname
        killwaves Ptmp
End

function spikekiller(inwave,n)
        Wave inwave
        variable n
        variable ind
        variable spikeRow, spikeCol
        for(ind=0;ind<n;ind+=1)
              imagestats/q /G={pcsr(a),pcsr(b),qcsr(a),qcsr(b)} inwave
              spikeRow=V_maxRowLoc
              spikeCol=V_maxColLoc
              Make/o /d/n=(5,5) tmp
              tmp=inwave[spikeRow-2+p][spikeCol-2+q]
              duplicate/o tmp tmp2
              imagestats tmp
              //print "avg=",V_avg,"\trms=",V_rms,"\tsdev=",V_sdev
```

```
            if (tmp[1][2]-tmp[1][1]>=V_sdev||tmp[1][2]-
tmp[1][3]>=V_sdev)
                  print "true -1"
                  tmp2[1][2]=(tmp[1][1]+tmp[1][3])/2

            endif
            if (tmp[3][2]-tmp[3][1]>=V_sdev||tmp[3][2]-
tmp[3][3]>=V_sdev)
                  print "true +1"
                  tmp2[3][2]=(tmp[3][1]+tmp[3][3])/2

            endif
            print "[",spikeRow,"][",spikeCol,"]=",V_max
            tmp2[2][2]=nan
            matrixfilter/n=5 NanZapMedian tmp2
            inwave[spikeRow][spikeCol]=tmp2[2][2]
      endfor
      //print tmp[2][2]
      killwaves tmp,tmp2

End

Function applymask(inwave)
      wave inwave
      variable w1,w2, ds
      w1=dimsize(inwave,0)/16
      w2=w1/2
      ds=dimsize(inwave,0)
      inwave[][(ds-1-w1),]=inwave[p][q]*exp(-((q-(ds-1-w1))/w2)^2)
      inwave[,w2][]=inwave[p][q]*exp(-((p-w2)/(w2/2))^2)
      inwave[(ds-1-w2),][]=inwave[p][q]*exp(-((p-(ds-1-w2))/(w2/2))^2)
End

outwave[][(1023-64),]=outwave[p][q]*exp(-((q-(1023-64))/32)^2)
outwave[,32][]=outwave[p][q]*exp(-((p-32)/16)^2)
outwave[(1023-32),][]=outwave[p][q]*exp(-((p-(1023-32))/16)^2)

Function gauss_sum4(w,x) : FitFunc
      Wave w
      Variable x

      //CurveFitDialog/ These comments were created by the Curve
Fitting dialog. Altering them will
      //CurveFitDialog/ make the function less convenient to work with
in the Curve Fitting dialog.
      //CurveFitDialog/ Equation:
      //CurveFitDialog/ f(x) = y0+A1*exp(-((x-x1)/w1)^2)+A2*exp(-((x-
x2)/w2)^2)+A3*exp(-((x-x3)/w3)^2)+A4*exp(-((x-x4)/w4)^2)
      //CurveFitDialog/ End of Equation
      //CurveFitDialog/ Independent Variables 1
      //CurveFitDialog/ x
      //CurveFitDialog/ Coefficients 13
      //CurveFitDialog/ w[0] = w1
      //CurveFitDialog/ w[1] = w2
      //CurveFitDialog/ w[2] = w3
      //CurveFitDialog/ w[3] = w4
      //CurveFitDialog/ w[4] = x1
      //CurveFitDialog/ w[5] = x2
```

```
	//CurveFitDialog/ w[6] = x3
	//CurveFitDialog/ w[7] = x4
	//CurveFitDialog/ w[8] = A1
	//CurveFitDialog/ w[9] = A2
	//CurveFitDialog/ w[10] = A3
	//CurveFitDialog/ w[11] = A4
	//CurveFitDialog/ w[12] = y0

	return w[12]+w[8]*exp(-((w[4]-x)/w[0])^2)+w[9]*exp(-((w[5]-
x)/w[1])^2)+w[10]*exp(-((w[6]-x)/w[2])^2)+w[11]*exp(-((w[7]-x)/w[3])^2)
End
Function gauss_sum(w,x) : FitFunc
	Wave w
	Variable x

	//CurveFitDialog/ These comments were created by the Curve
Fitting dialog. Altering them will
	//CurveFitDialog/ make the function less convenient to work with
in the Curve Fitting dialog.
	//CurveFitDialog/ Equation:
	//CurveFitDialog/ f(x) = y0+A1*exp(-((x-x1)/w1)^2)+A2*exp(-((x-
x2)/w2)^2)
	//CurveFitDialog/ End of Equation
	//CurveFitDialog/ Independent Variables 1
	//CurveFitDialog/ x
	//CurveFitDialog/ Coefficients 7
	//CurveFitDialog/ w[0] = w1
	//CurveFitDialog/ w[1] = w2
	//CurveFitDialog/ w[2] = x1
	//CurveFitDialog/ w[3] = x2
	//CurveFitDialog/ w[4] = A1
	//CurveFitDialog/ w[5] = A2
	//CurveFitDialog/ w[6] = y0

	return w[6]+w[4]*exp(-((w[2]-x)/w[0])^2)+w[5]*exp(-((w[3]-
x)/w[1])^2)
End
Function gauss_one(w,x) : FitFunc
	Wave w
	Variable x

	//CurveFitDialog/ These comments were created by the Curve
Fitting dialog. Altering them will
	//CurveFitDialog/ make the function less convenient to work with
in the Curve Fitting dialog.
	//CurveFitDialog/ Equation:
	//CurveFitDialog/ f(x) = y0+A1*exp(-((x-x1)/w1)^2)
	//CurveFitDialog/ End of Equation
	//CurveFitDialog/ Independent Variables 1
	//CurveFitDialog/ x
	//CurveFitDialog/ Coefficients 4
	//CurveFitDialog/ w[0] = w1

	//CurveFitDialog/ w[1] = x1

	//CurveFitDialog/ w[2] = A1

	//CurveFitDialog/ w[3] = y0
```

```
	return w[3]+w[2]*exp(-((w[1]-x)/w[0])^2)
End
Function timeIntegrate(inWave)
	Wave inWave
	duplicate/o inWave intSpec
	Redimension/N=-1 intSpec
	variable i
	intSpec=0
	for(i=0;i<dimsize(inWave,1);i+=1)
		intSpec=inWave[p][i]+intSpec[p]
	endfor
End
Function freqIntegrate(inWave)
	Wave inWave
	make/o /d/n=(dimsize(inwave,1)) intIntensity
	variable i
	intIntensity=0
	for(i=pcsr(A);i<pcsr(B);i+=1)
		intIntensity=inWave[i][p]+intIntensity[p]
	endfor
	setscale /P x dimoffset(inwave,1), dimdelta(inwave,1),
intIntensity
End
function matrixfitter(matrixwave,xwave,curve,start)
	//fits the frequency-time data from a Sum frequency XFROG
experiment
	//matrixwave is the name of the 2-D data wave
	//xwave is the name of the wave containing the frequency points
(converted from wavelength)
	//curve is 1 or 2 to use either single or double gaussian fit
	//start is the index of the time step to start from, it is good
to skip the time steps before the start of the fluorescence
	//To begin, use the built-in fitting dialog to fit at least one
spetrum from the data set. This sets W_coef to something that
	//the matrix fitter can use as an intial guess for fitting the
rest of the time steps.
	wave matrixwave,xwave
	variable curve,start
	wave T_Constraints, W_coef, W_sigma
	variable rows, cols, n, pms
	variable V_FitError
	duplicate/o matrixwave tmp, params
	duplicate/o W_coef coef_Start
	rows=dimsize(tmp,0)
	cols=dimsize (tmp,1)
	pms=dimsize (W_coef,0)
	Redimension/N=((2*pms+1),-1) params
	params[][,start-1]=0
	make/o /n=(rows) tmp2
	//W_coef=coef_Start
	if(curve==2)
		for(n=start;n<cols;n+=1)
			tmp2=tmp[p][n]
			V_FitError=0
			FuncFit /Q gauss_sum W_coef  tmp2[pcsr(A),pcsr(B)]
/X=xwave /C=T_Constraints
			if (V_FitError!=0)
```

```
                        W_coef=coef_Start
                        FuncFit /Q gauss_sum W_coef
tmp2[pcsr(A),pcsr(B)] /X=xwave  ///C=T_Constraints
                  endif
                  if (V_FitError==0)
                        params[,(pms-1)][n]=W_coef[p]
                        params[pms,(2*pms-1)][n]=W_sigma[p-pms]
                        params[(2*pms)][n]=V_chisq
                  else
                        params[,(pms-1)][n]=Nan
                        params[pms,(2*pms-1)][n]=nan
                        params[(2*pms)][n]=nan

                  endif
            endfor
      else
            for(n=start;n<cols;n+=1)
                  tmp2=tmp[p][n]
                  V_FitError=0
                  FuncFit /Q gauss_one W_coef  tmp2[pcsr(A),pcsr(B)]
/X=xwave ///C=T_Constraints
                  if (V_FitError!=0)
                        W_coef=coef_Start
                        FuncFit /Q gauss_one W_coef
tmp2[pcsr(A),pcsr(B)] /X=xwave ///C=T_Constraints
                  endif
                  if (V_FitError==0)
                        params[,(pms-1)][n]=W_coef[p]
                        params[pms,(2*pms-1)][n]=W_sigma[p-pms]
                        params[(2*pms)][n]=V_chisq
                  else
                        params[,(pms-1)][n]=Nan
                        params[pms,(2*pms-1)][n]=nan
                        params[(2*pms)][n]=nan
                  endif
            endfor

      endif

      killwaves tmp, tmp2
end

function mkfreqcm(inwave)
      wave inwave
      make/o/d /n=(dimsize(inwave,0)) freqcm
      copyscales inwave, freqcm
      freqcm=1e7/x
end
function mkfreqHz(inwave)
      wave inwave
      make/o/d /n=(dimsize(inwave,0)) freqHz
      copyscales inwave, freqHz
      freqHz=299792458e9/x
end
function mktestspec(inwave,funits)
      wave inwave
      string funits
      make/o/d /n=(dimsize(inwave,0)) testspec
```

```
	testspec=inwave[p][qcsr(A)]
	variable/g firstspecPos=qcsr(A)
	if(stringmatch (funits,"hz")|stringmatch (funits,"Hz"))
		mkfreqHz(inwave)
		display testspec vs freqHz
	else
		mkfreqcm(inwave)
		display testspec vs freqcm
	endif
end
Function polyoffset(w,x) : FitFunc
	Wave w
	Variable x

	//CurveFitDialog/ These comments were created by the Curve
Fitting dialog. Altering them will
	//CurveFitDialog/ make the function less convenient to work with
in the Curve Fitting dialog.
	//CurveFitDialog/ Equation:
	//CurveFitDialog/ f(x) = k0+k1*(x-x0)+k2*(x-x0)^2+k3*(x-x0)^3
	//CurveFitDialog/ End of Equation
	//CurveFitDialog/ Independent Variables 1
	//CurveFitDialog/ x
	//CurveFitDialog/ Coefficients 5
	//CurveFitDialog/ w[0] = k0
	//CurveFitDialog/ w[1] = k1
	//CurveFitDialog/ w[2] = k2
	//CurveFitDialog/ w[3] = k3
	//CurveFitDialog/ w[4] = x0

	return w[0]+w[1]*(x-w[4])+w[2]*(x-w[4])^2+w[3]*(x-w[4])^3
End

Function avgfrequency(inwave, twind)
	wave/c inwave
	variable twind //in fs
	duplicate/o inwave tmp1, tmp2,avgfreq
	redimension/r tmp1, tmp2,avgfreq
	redimension /n=(dimsize(inwave,0)-
trunc(twind/dimdelta(inwave,0))),avgfreq
	//print dimsize(inwave,0)-trunc(twind/dimdelta(inwave,0))
	avgfreq=0
	tmp1=-atan2(imag(inwave),real(inwave))
	tmp1+=pi
	unwrap 2*pi, tmp1
	tmp1-=pi

	Differentiate tmp1
	tmp2=magsqr(inwave[p])
	tmp1*=tmp2[p]
	avgfreq=area(tmp1,pnt2x(tmp1,p),pnt2x(tmp1,p)+twind)//(trunc(twin
d/dimdelta(inwave,0))*dimdelta(inwave,0))
//area(tmp2,pnt2x(tmp1,p),pnt2x(tmp1,p)+twind)
	variable tmindex
	//for (tmindex=0; tmindex<dimsize(inwave,0)-
trunc(twind/dimoffset(inwave,0)); tmindex+=1)
```

```
	//avgfreq[tmindex]=area(tmp1,pnt2x(tmp1,tmindex),pnt2x(tmp1,tmind
ex)+twind)
	//endfor
	killwaves tmp1,tmp2
End

Function avgfrequency2(inwave, twind)
	wave/c inwave
	variable twind //in fs
	duplicate/o inwave avgfreq2
	redimension/r avgfreq2
	redimension /n=(dimsize(inwave,0)-
trunc(twind/dimdelta(inwave,0))),avgfreq2
	//print dimsize(inwave,0)-trunc(twind/dimdelta(inwave,0))
	//print trunc(twind/dimdelta(inwave,0))
	avgfreq2=0
	variable tmindex
	for (tmindex=0; tmindex<dimsize(inwave,0)-
round(twind/dimdelta(inwave,0)); tmindex+=1)

	FFT/OUT=4/PAD={dimsize(inwave,0)}/WINF=Hanning/RP=[tmindex,tminde
x+round(twind/dimdelta(inwave,0))]/DEST=tmp1 inwave

	FFT/OUT=4/PAD={dimsize(inwave,0)}/RP=[tmindex,tmindex+round(twind
/dimdelta(inwave,0))]/DEST=tmp1 inwave
		Integrate tmp1 /D=tmp2
		tmp1*=x
		Integrate tmp1
		avgfreq2[tmindex]=tmp1[dimsize(inwave,0)-
1]//tmp2[dimsize(inwave,0)-1]

	endfor
	killwaves tmp1,tmp2
End

Function avgfrequencyChg(inwave, twind)
	wave/c inwave
	variable twind //in fs
	duplicate/o inwave tmp1, tmp2,avgfreq
	redimension/r tmp1, tmp2,avgfreq
	redimension /n=(dimsize(inwave,0)-
trunc(twind/dimdelta(inwave,0))),avgfreq
	//print dimsize(inwave,0)-trunc(twind/dimdelta(inwave,0))
	avgfreq=0
	tmp1=-atan2(imag(inwave),real(inwave))
	tmp1+=pi
	unwrap 2*pi, tmp1
	tmp1-=pi

	Differentiate tmp1
	//Differentiate tmp1
	tmp2=magsqr(inwave[p])
	tmp1*=tmp2[p]
	//tmp1*=cabs(inwave[p])
	avgfreq=area(tmp1,pnt2x(tmp1,p),pnt2x(tmp1,p)+twind)/(trunc(twind
/dimdelta(inwave,0))*dimdelta(inwave,0))
//area(tmp2,pnt2x(tmp1,p),pnt2x(tmp1,p)+twind)
```

```
	variable tmindex
	//for (tmindex=0; tmindex<dimsize(inwave,0)-
trunc(twind/dimoffset(inwave,0)); tmindex+=1)

	//avgfreq[tmindex]=area(tmp1,pnt2x(tmp1,tmindex),pnt2x(tmp1,tmind
ex)+twind)
	//endfor
	killwaves tmp1,tmp2
End

Function avgFromGaussSum(params,xwave)
	wave params,xwave
	variable curve,start
	variable rows, cols, n, normConst
	rows=dimsize(params,0)
	cols=dimsize (params,1)
	duplicate/o xwave  tmpspec
	make/o /d/n=7 pms


	make /o/d /n=(cols) tmp2, tmp3
	SetScale/I x (xwave[dimsize(xwave,0)-1]),(xwave[0]),"", tmpspec
	duplicate /o tmpspec tmp
	for(n=0;n<cols;n+=1)
		pms[0,5]=params[p][n]
		pms[6]=0
		//tmp=gauss_sum(pms,xwave[p])*xwave[p]
		//integrate tmp /X=xwave
		//avgGaussFreq[n]=tmp[dimsize(xwave,0)-1]
		tmpspec=gauss_sum(pms,x)*x
		tmp=gauss_sum(pms,x)
		integrate tmp
		integrate tmpspec
		//avgGaussFreq[n]=sum(tmpspec,xwave[255],xwave[0])
		//avgGaussFreq[n]=area(tmpspec) //integrate tmpspec
		tmp2[n]=tmpspec[dimsize(xwave,0)-1]/tmp[dimsize(xwave,0)-1]
		tmp3[n]=tmp[dimsize(xwave,0)-1]
		//tmp3[n]=sum(tmp,xwave[255],xwave[0])
	endfor

	SetScale/P x dimoffset(params,1),dimdelta(params,1),"",tmp2,tmp3
	string newname= nameofwave(params)+"aGaussV"
	duplicate/o tmp2, $newname
	newname= nameofwave(params)+"IntInt"
	duplicate/o tmp3, $newname
	//avgGaussFreq/=(xwave[0]-xwave[255])
	//print area(xwave)
	killwaves pms, tmpspec, tmp2, tmp3, tmp
end
Function avgFromGauss_sum4(params,xwave)
	wave params,xwave
	variable curve,start
	variable rows, cols, n, normConst
	rows=dimsize(params,0)
	cols=dimsize (params,1)
	duplicate/o xwave  tmpspec
	make/o /d/n=15 pms
```

```
        make /o/d /n=(cols) tmp2, tmp3
        SetScale/I x (xwave[dimsize(xwave,0)-1]),(xwave[0]),"", tmpspec
        duplicate /o tmpspec tmp
        for(n=0;n<cols;n+=1)
              pms[0,11]=params[p][n]
              pms[12]=0
              //tmp=gauss_sum(pms,xwave[p])*xwave[p]
              //integrate tmp /X=xwave
              //avgGaussFreq[n]=tmp[dimsize(xwave,0)-1]
              tmpspec=gauss_sum4(pms,x)*x
              tmp=gauss_sum4(pms,x)
              integrate tmp
              integrate tmpspec
              //avgGaussFreq[n]=sum(tmpspec,xwave[255],xwave[0])
              //avgGaussFreq[n]=area(tmpspec) //integrate tmpspec
              tmp2[n]=tmpspec[dimsize(xwave,0)-1]/tmp[dimsize(xwave,0)-1]
              tmp3[n]=tmp[dimsize(xwave,0)-1]
              //tmp3[n]=sum(tmp,xwave[255],xwave[0])
        endfor

        SetScale/P x dimoffset(params,1),dimdelta(params,1),"",tmp2,tmp3
        string newname= nameofwave(params)+"aGaussV"
        duplicate/o tmp2, $newname
        newname= nameofwave(params)+"IntInt"
        duplicate/o tmp3, $newname
        //avgGaussFreq/=(xwave[0]-xwave[255])
        //print area(xwave)
        killwaves pms, tmpspec, tmp2, tmp3, tmp
end
```

Appendix C

Collected instructions for data analysis in Igor Pro

Loading data from the XGROG instrument into Igor Pro

Open Igor and look for the XFROG menu.

From that menu choose "Load XFROG Data".

Click continue in the first window, then find the folder where you put the two data files and click open or choose.

The next window should read "Enter X Number of Serial Points". The number in the entry field should read 256 and you can click continue. If it doesn't, the data file has been damaged and you have to correct it.

The next window should read "Enter Time-Wavelength Matrix Base Name". This is just base name for waves to store each of the data files that you are about to load. I usually use a short base like "r" for run, but you can choose whatever you wish. Click continue.

Next, in the history area you will see a bunch of numbers scrolling by. The numbers should be the time step of each spectrum from each of the data files. Check to make sure that the file has loaded correctly. The numbers in the each of the 2-D Time-Wavelength

waves should be between about 500 and 66000. You can use Igor's built in Data Browser to check things out.

Background subtraction

First you should make an image plot of the 2-D time-frequency data. Windows>>New>>Image Plot... Select one of your data sets R0, R1, etc. for the Z data and "_calculated" for X and Y. Choose a color scheme in the "Appearance" window. I usually go with Rainbow because it reveals subtle features in the intensity. While in the Appearance window also adjust the "first Z at" to 600 and the "last Z at" to 800. Then click Do It.

Next, you'll need to get rid of the spikes. If you want to adjust the color scale again you can right click if you've got a two button or command click on the graph to bring up the contextual menu and then change the first and last Z values. Next, expand the image plot and then type command-i. This will add a couple of cursors to the plot. Look for spikes which will be high z values that only exist in 1-2 adjacent pixels and place the round cursor to the lower left of the spike and then the square cursor to the upper right. This defines a box for the "Spike Killer" operation. The spike killer looks inside the box and finds pixels that are more than one or two sigma higher than the average of the box and replaces then with the average of the 24 nearest neighbors. To activate the spike killer, go to the command line and type spikekiller(data,n) where you replace data with the name of your time-frequency data ("R0", for instance) and n with the number of passes you want

the spike killer to make on the box you just defined. This usually the number of spikes in the box. Then hit return. Two things are important to use this, first that you have defined the box properly by having the circle cursor in the lower left hand (or smaller indices) and the square cursor in the upper corner (or higher indices). The other important thing is that the plot with the cursors be the topmost plot or "target plot" in Igor speak. Move the cursors around and enter the spikekiller command until you get rid of all of the spikes.

Now you are ready to subtract a background. I've found that using the first couple of time steps of the data set works the best, if they are truly before t=0 of course. There are lots of ways to do this. Here's the the method that I've settled on. Make a background wave by typing

Make /n=256 background

Here the /n=256 flag is the number of frequency or wavelength points you have in your data (should be 256). "background" is just the name of the new wave. Assuming that you are working on a time frequency data set called "R0" and the first 4 time steps are completely free of any pump or fluorescence upconversion signal this is how to make the background wave.

In the command window:

background=(R0[p][0]+R0[p][1]+R0[p][2]+R0[p][3])/4

The notation R0[p][0], R0[p][1],... means the 0th, 1st, ... column of R0. So we are just averaging the first 4 columns or time steps of R0.

Next, I'd duplicate the original data.

In the command window:

Duplicate R0 Bob

"Bob" is the name of the copy of R0.

In the command window:

Bob=R0[p][q]-background[p]

Now make an Image Plot of Bob in the same way that you did with R0, but set the first and last Z values to 0 and ~150 instead of 600 and 800. Next you can plot individual spectra by going to Windows>>New Graph and make sure you have selected the "More Choices" option. Select Bob for the Y data and "_calculated" for X data, then click "Add". In the lower part of the New Graph dialog you will see Bob and two boxes each inside square brackets. The first set of brackets specifies the range of x values and the second specifies the y range. Change the value in the y brackets to something like 30. This will correspond to the spectrum at the 31st time step of the data set. Then click do it.

Add additional spectra with Graphs>>Append traces to Graph and follow the steps just above, but choose a different time step. You can modify the color and style of the traces by double clicking on the trace you want to modify. You can rescale or modify the axis by double clicking on them.

To export to graphs to Power Point you can just copy and paste or use the "Export Graphics" dialog in the Edit Menu. The export graphics dialog also copies the graph to the clipboard for pasting. You can save graphs using the
"Save graphics" dialog in the File menu.

Setting the scale of the XFROG data

To set the scale you need to know what the monochromator was set to when you took the data. Then make sure you have your nice image plot. Then, under the XFROG menu choose "open XFROG panel". This will open a new panel with some tool for XFROG data. Choose your data from the list that is labeled FROG trace. If you don't have your gate pulses loaded in, skip that line and go to the next that says "experimental time step/fs". Use the arrows to change the value, or you can enter it manually, to whatever your time steps were. Then enter the monochromator setting in the next box. Leave the last box alone. Then click "Set Scale" this will automatically change the scale of your XFROG trace to real units of nm and fs.

Peak Fitting to a Sum of Gaussians to get peak frequency vs time step information

Make sure your XFROG trace (image) is the topmost window and goto the "Graph" menu and select "Show Info". This will put two cursors (a circle and a square) on the bottom of the graph. Drag the circle onto the graph and place it at the start of your upconverted fluorescence signal. The upconverted fluorescence signal looks like a finger, just place the circle cursor at the tip of the finger. Next go down to the command window and type

mktestspec("your data","cm")

but replace "your data" with the name of your XFROG trace. (make sure that you type the quotes in the second argument like this "cm") This will make a plot of the spectrum at the time step were you put the cursor. It copies the spectrum from the trace (a 2-d wave) and puts it into a new wave (1-d) called testspec.***(see the note below). Hopefully you will see at least a little bump where your data should be. If not close that new plot and move the cursor to a new point a little later in the trace and execute the command again.

Now with the new plot on top, select "Show Info" from the Graph menu. This will put the two cursors on the new plot. Put the cursors on the plot so that they encompass the the spectrum. Try to enclose 10-20 points on each side of the fluorescence but make sure that you don't encompass the cross-correlation signal. Next select "Curve Fitting" from the

Analysis menu. Select "gauss_one" for one gaussian or "gauss_sum" for a sum of two gaussians and click the "From Target" box. Select testspec for "Y data" , then the "X data" should be automatically set to freqcm (if not, select it manually). Click on the Data Options tab and then click the "cursors" button. Click the Coefficients tab. Enter initial guesses for all of the parameters (there are 4 for the gauss_one and 7 for the gauss_sum), don't enter anything for epsilon or constraints and don't hold anything. However, select "From Coefficients List" from the Constraints itemt on the right side. The "w"s are widths, the "x"s are peak positions, the "A"s are amplitudes and yo is a baseline offset. To see how close your guesses are, click "Graph Now". This will graph your initial guesses. Refine your guesses until they are kind of close, then hit the "Do It" button. This should fit this test spectrum. If it says there was an error of some sort, go back and refine your initial guesses (because you suck and your mama wear's a thong and combat boots).

Now to fit all of the spectra in the XFROG trace.

Make sure that the testspec vs freqcm plot is on the top and still has the circle and square cursors surrounding the upconversion spectrum. Then type the following in the command window.

matrixfitter("your data",freqcm,"curve",firstspecPos)

replace "your data" with the name of your XFROG trace and "curve" with either 1 for gauss_one or 2 for gauss_sum (this _HAS TO_ match the curve you used to fit testspec). The matrixfitter will crank through all of the spectra in the XFROG trace and record the value and standard deviation of each parameter and the chi-squared for each fit. The

matrixfitter fucntion will create a new matrix called params which contains all of this info. For instance, if you chose gauss_sum then params will have 15 rows and N columns where N is the number spectra, or time steps, in the XFROG trace. Row 0 would have the width of the first gaussian, row 1 would have the second width. Next are the peak frequencies, then the amplitudes then the row with the baseline offset. Next, are the sigmas for each parameter respectively. So the sigma for the first width (row 0) would be in row 7 and so on. The last row contains the chi-squared for the fit.

If the matrixfitter chokes on a spectrum and produces an error, it will record a list of "nan"s, or not a number, for the parameters for that time step. These will just be blank entries the wave and will look like missing values in the params matrix. If you get a lot of missing parameters, then I'd go back and re-fit the testspec and also check to make sure that cross correlation signal is not within the cursors.

***If you move the cursor and execute the command again with the trace in the topmost window, you will get a new plot with the new spectrum that is under the cursor. However, your old plot will also have the new spectrum because the function overwrote the data that was previously in the wave called "testspec".

Setting the limits and resampling of the XFROG data to use Phase Retrival Analysis

With the smoothed and background subtracted XFROG data selected as the top graph, select "Show Info" from the Graph menu if cursors are not already on the graph. Then place the cursors on the high and low limits of the SFG fluorescence signal. Next bring the "X-FROG Tools" panel forward and click the button marked as "Get Limits". The wavelength values of the two cursors should appear in the "start" and "end" displays. Next, choose the number of grid points you wish to have in the final resampled XFROG trace by using the up and down arrows next to the lable "Grid Dimension". This number should be an even multiple of 128, such as 128, 256, 512, 1024, 2048, etc. Then click "Resample".

Appendix D

Upconversion Instrument Manual Including instructions for Prism Compression

Introduction:

When beginning a new round of gated fluorescence data collection, it is advantageous to tune the entire system including the oscillator and the prism pairs. It will also be necessary to follow this procedure after there has been a large change in temperature in the lab or if the laser table is disturbed in some way (such as running out of nitrogen to supply the support legs). The red beam prism compressor can be adjusted using the FemoChorom autocorrelator. The blue prism compressor will have to be adjusted using the red-blue cross-correlation signal.

Procedures:

First, set up the Ocean Optics emission spectrometer to collect a small portion of the oscillator emission. This can be done by picking off a small portion of the beam with a microscope slide stuck to an optical mount. Try to position the emission collector so that the sampled beam is entering the optical fiber chuck as straight as possible. It will be necessary to provide a target for the sampled beam to be dispersed before it is collected by the spectrometer. I have found that 2-3 layers of lens tissue are sufficient. The tissue is

placed directly in front of the fiber chuck/collector. Another suitable dispersion target is a piece of matte (not glossy) Scotch tape on a microscope slide. The idea is to scatter the laser emission with out absorption. The CW beam will saturate the spectrometer. If the spectrometer is still saturated after mode locking, some neutral density filters in front of the dispersion target will nock the intensity down a bit. (ND filters appear grey and attenuate all wavelengths equally) A power meter should be positioned after the microscope slide beam splitter to measure the beam power. Note that the microscope slide beam splitter will decrease the power of the transmitted beam by about 10%. With this tandem set up, collection of the beam spectrum and measurement of the power can be done simultaneously and the oscillator can be tweaked for the desired bandwidth. For solvation dynamics experiments, a FWHM bandwidth of ~60 nm @800 nm is sufficient. This bandwidth will result in a transform limited pulse of ~16 fs. At this bandwidth, it should be possible to get a mode locked power of at least 540 mW without much tweaking. Make a note of the center wavelength of the laser emission.

The procedure for tweaking up the KM Labs Ti:sapp oscillator is covered it its operation manual.

Next the doubling crystal and associated optics need to be optimized for peak blue power. Remove the beam splitter and power meter. Make sure the beam passes through the center of the doubling crystal focusing lens and the doubling crystal itself. If it is off center, use the preceding two steering mirrors to correct it. Place the power meter in the blue beam path and set the range to 30 mW. Using the longitudinal adjustment on the lens before the doubling crystal and the vertical tilt of the crystal holder, maximize the blue power. Then, use the blue mirror on the flipper mount to direct the blue beam

through the center of the blue collimation lens. Make sure the blue beam is not being clipped by the waveplate after the collimation lens. If it is clipped, move the waveplate base to eliminate the clipping.

The blue beam should graze cleanly over the square compressor pick off mirror and strike the first and second prisms a few millimeters from the apex. Next check the alignment of the exiting compressed beam. Use a card near the pick off mirror to align the exiting beam directly below the entering beam. If the exiting beam is not directly below, use the horizontal adjustment of mirror after the second prism to steer the lower beam to achieve horizontal alignment. Use the vertical adjustment of that same mirror to adjust the height of the exiting beam to be as close to the top edge of the pick off mirror as possible without clipping.

The red beam should also cleanly graze the red pick off mirror. Use the same procedure as the blue beam to properly align the red beam into and out of the red beam prism compressor.

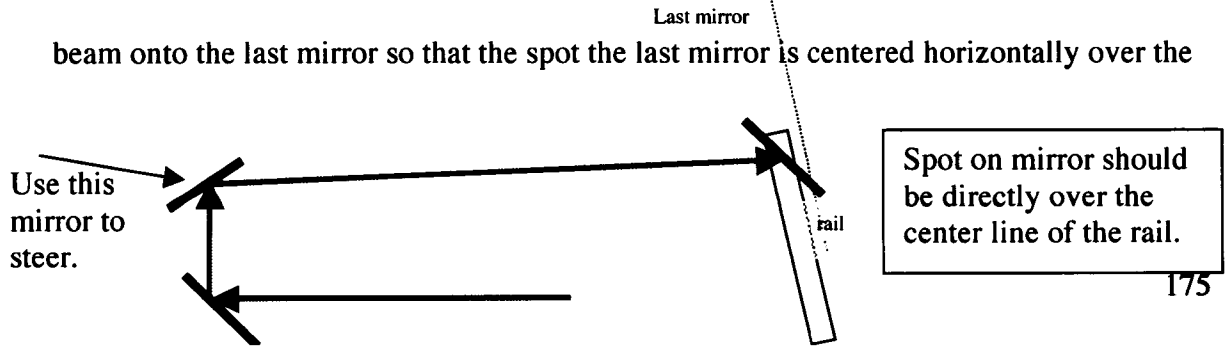
Use a "correct height" card of 4 inches to correct the height of the blue and red beams after they have been compressed. The beams should be set the correct height after the first turning mirror following the respective pick off mirrors. This is accomplished by setting the card near the first turning mirror and adjusting the spot height with the pick off mirror. Then the card is moved away from the turning mirror and the turning mirror vertical tilt is adjusted to bring the spot on the card to the correct height. Next, ensure that the beams are incident on the clear apperature of remaining mirrors that steer the beams into the upconversion optics.

Ideally to adjust the red prism pair insertion, you need to sample the beam just after the lens that focuses the red beam onto the mixing crystal. Due to the geometry of the experiment this is somewhat difficult, so you can sample right before the lens and right after the waveplate. Use metal mirrors (gold or silver) to sample the beam and direct it to the autocorrelator. Make sure that the beam is reflecting off of the metal side of the mirrors and not the back. It may take 3 or 4 mirrors to get the beam steered into the autocorrelator. The autocorrelator requires vertically polarized light to function efficiently. The polarization of the beam after the waveplate should already be set to vertical so no further change will be necessary. (If the beam is sampled before the waveplate, the polarization should be adjusted to vertical.)

The procedure for using the Femtochrome autocorrelator is covered in its operation manual.

Using the micrometers on the prism stages, adjust the amount of prism inserted into the beam path to achieve the narrowest pulse width. It may take several iterations of optimizing one prism, then the other. At this point, the compressor has been optimized to compensate for all of the optics except the final focusing lens. To compensate for this lens you can decrease the amount of prism insertion by 1-2 millimeters on each prism as an estimate. As before, make sure that the beams going into and out of the prism pair are horizontally aligned.

Make sure that the beam height is correct up until the last mirror. Next, align the blue beam into the flow cell. To do this, use the second to last steering mirror to steer the beam onto the last mirror so that the spot the last mirror is centered horizontally over the



rail and last mirror mount.

Now use the last mirror to direct the beam through the horizontal center of the blue focusing lens. This should result in a spot that is slightly off center on the flow cell. The spot should be displaced about 1-2 spot diameters from the center of the flow cell in the direction of the mixing crystal (right of center when standing behind the elliptical reflector). Pitch the beam downward slightly with the last mirror, but not too much. The beam should pass through the lens at just under 4 inches (3.9, or so). If at this point the beam is not centered vertically on the lens, adjust the height of the lens so that the beam is centered.

To correctly position the reflector, begin by standing a card on top of the mixing crystal rotation stage and against the front side of the mixing crystal holder. Then use the three ordinal controls on the elliptical reflector mount to produce the smallest, roundest fluorescence image on the card. After obtaining a small round fluorescence image, remove the card and ensure that the fluorescence is cleanly imaged on the mixing crystal. If the image is on the edge of the crystal, try steering the blue beam to the left or right with the last turning mirror, then repeat the reflector optimization procedure. In the past this procedure produced a fluorescence image that was just to the right of center of the mixing crystal when viewed from the front. This image will serve as a target for the gate pulse.

To position the red beam start by assuring that it is at the correct height starting from the first mirror after the compressor pick off mirror. Use the correct height card procedure mentioned above for this. Next, trace the positions of the red beam focusing lens and waveplate mounts with a Sharpie before removing them. Also note which face

of the mirror the beam enters. (There should be a label on the lens mount with an arrow indicating the beam direction.) Using the last 1" mirror, direct the red beam to the small gold turning mirror just before the mixing crystal. Use the small gold mirror to overlap the red beam image with the fluorescence image. At this point the red beam image will be several times larger than the fluorescence image. Make sure that the fluorescence image is at the center of the red image. Then replace the waveplate using your trace as a guide. Position it so that the overlap of the two images on the crystal is unaffected. Now, replace the focusing lens also using the trace as a guide. The mount for the focusing lens should butt against the gold mirror mount. Adjust the horizontal position of the lens and mount so that the focused red spot is at the center of the fluorescence image on the crystal. Use the back reflection of the red beam to assure that the lens is normal to the red beam. The height of the lens may also need to be adjusted so that the lens isn't walking the beam vertically.

With the gate and fluorescence images overlapped on the crystal, you can start to look for upconversion signal. Make sure that the red pulse delay stage is in "positive time". Most of the time this can be accomplished by simply moving the delay by 500 ps or so in the positive direction. Next, make sure that the turning mirror is in place inside of the monochromator. This is the mirror that steers the emission into the PMT. Open the entrance slits to the monochromator as far as possible and make sure that the PMT shutter is open. Set the monochromator to 315 nm. Make sure that the red beam polarization is set to vertical and the blue beam is set to Magic Angle (54.7° away from vertical).

To align the mirrors that take upconverted signal into the monochromator, tune the mixing crystal for maximum blue power. This should be about 45° or so. Next, adjust

the position of the upconversion collection lens with the micrometer so that both the green fluorescence and some of the doubled red emission are reflected by the first turning mirror. The upconversion signal will follow a path that is somewhere between that of the green and blue. Use a card to observe the green and blue images on the entrance slits of the monochromator. The blue should be on the left and the green should be on the right. At this point it is most important that the vertical position of the images is correct on the slits. To optimize the vertical position, adjust the horizontal tilt of the first turning mirror so that some of the blue is entering the monochromator. This should produce several hundred counts on the photon counter. Using the adjustments on the two upconversion signal turning mirrors, peak up on the blue counts. Optimizing on the doubled red signal will get the vertical alignment close enough so that only a moderate horizontal adjustment of the first turning mirror will result in upconversion signal. Before making the horizontal adjustment to steer the upconversion signal into the monochromator, change the crystal angle to about 54°. Then, using the horizontal adjustment on the first upconversion turning mirror, move the images of the blue and green on the slits to the left while watching the counts on the photon counter. The counts should decrease as you steer the blue image away from the slit, and then increase as the upconversion signal enters the monochromator. It may take as many as 5-7 complete revolutions of the horizontal tilt on the first mirror to see upconversion signal. This method usually produces 200-400 counts above background initially. To check for signal, block the blue excitation pulse before the blue beam focusing lens, then block the red before the focusing lens. If you see the suspected signal decrease to background after each beam block then this is most likely upconversion signal. If blocking a beam has no effect on the count level then this is not

upconversion signal. After you have verified that the PMT is detecting upconversion signal, peak up the counts using the 3 ordinal adjustments of the elliptical reflector, the crystal angle, the collection lens, and the two signal turning mirrors. The focal points of the red and blue focusing lenses can also be adjusted slightly. It should be fairly easy to get 2000-3000 counts using the C153/MeOH alignment solution and the Cleveland Crystals BBO (white holder with blue lettering). If no signal is seen, use the adjustments on the small gold turning mirror to improve the overlap of the red and fluorescence images on the crystal and repeat the above procedures starting from optimizing the doubled red signal.

After the upconversion signal has been optimized, time zero delay position must be found. This can be done my moving the delay stage in the negative direction and looking for the signal on the photon counter to drop significantly. After the zero delay position has been determined, move the delay stage to about 5 ps. Then, move the monochromator wavelength 5 to 10 nm to the blue. Adjust the crystal angle and signal turning mirrors to maximize the upconversion signal. Repeat this procedure until you have reached 295 nm on the monochromator. It is very probable that you will only see 100-200 counts above background after optimization at this wavelength. It might also be necessary to move the delay stage closer to the zero delay position to see any signal at this wavelength.

After optimizing for signal at 295 nm take a short scan using the single wavelength fluorescence decay Labview program. Then position the delay stage at the point of maximum signal according to the decay trace. Next, change the monochromator setting to 1/3 of the central oscillator emission wavelength you measured earlier with the

Ocean Optics spectrometer. For instance if you measured the wavelength of the laser output to be 810 nm, set the monochromator to 270 nm to detect the cross-correlation signal. Don't assume that the KM Labs laser will always produce 800 nm pulses-it is almost never centered at 800 nm. Now, rotate the crystal mount clockwise. You should see the counts on the photon counter increase dramatically as the crystal rotation stage approaches ~65-67°. At this point the cross-correlation is probably saturating the PMT. Close the slits (vertical and horizontal) until the photon counter reads no more than about 10,000,000 counts. Now go back to the fluorescence decay Labview program and run another short scan to find the maximum signal. Move the stage to the maximum signal and then optimize the crystal angle and the steering mirrors and adjust the slits if necessary to further attenuate the cross-correlation signal.

The cross-correlation signal will be used to adjust the prism glass insertion in the blue beam compressor. Start by turning the micrometer on the first blue prism one full turn to move the beam closer to the apex of the prism. This decreases the total amount of glass in the blue beam, which has the same effect as shortening the distance that the blue beam travels. Therefore, the zero delay position will move in the negative direction. One turn of the micrometer moves the prism stage by 0.5 mm, this in turns moves the zero delay position by about 2.5 ps. Repeat this procedure until the prism insertion has been optimized, then move to the second prism and repeat until it has been optimized. Then move back to the first and make sure it is still optimized. Before moving on to collect data, make sure that the blue and red beams are not being clipped by the edge of any of the prisms.

Appendix E

The list of measurements required for a good FROG experiment

(The order could be considered quasi chronological, but ultimately it is irrelevant.)

Collect emission spectra of gate pulse, pump, and emission (all not time resolved).

Time integrated emission spectra are crucial for the signal/phase retrieval analysis programs because they serve as an internal error check. The sum of all of the time resolved spectra retrieved from the FROG experiment (called the frequency marginal) should be the proportional to the time integrated emission spectrum. Since the time resolved spectra are essentially time slices of the integrated emission spectrum, their sum should be proportional to the integrated signal. The best emission spectra will be obtained using the pump pulse as the excitation source. The same arguments can be made to justify collection of the gate and pump pulse spectra. Collect all with the Ocean Optics spectrometer if possible. Collect FROG for gate pulse and X-FROG for pump.

In order to perform the phase retrieval of the emission signal, the electric field of the gate pulse is required. It is also a good idea to characterize the pump pulse to determine its duration and phase curve.

Collect FROG data out to at least 100 ps before changing any experimental parameter like the BBO crystal angle. The complete spectral evolution of the emission

signal needs to be measured for each set of experimental parameters for proper calculation of the frequency marginal.

Collect an entire spectrum.

This may require the lowest resolution grating in the spectrometer. The more I think about this, the more I think that the 600 grooves/mm grating should be used. For the reason mentioned above and also to increase the amount of signal per pixel.

Collect a good background.

A background that accounts for the exact experimental conditions is necessary because of the very low intensity signals that are measured. This background can be obtained by moving the delay stage to a position 10-50 ps before peak pump pulse intensity is detected.

Try to collect FROG data from a wide range of crystal angles.

Collecting data from a wide range of crystal angles will provide phase matching over a broad spectral range. This is necessary because the crystal thickness prevents simultaneous phase matching over the entire spectrum of the emission of a laser dye such as Coumarin 153 or 343.

Laser stuff:

It doesn't appear that using the shortest possible gate pulse is adventagous. It seems that a 40 fs gate pulse with a correspondingly good blue pulse should be sufficient.

In this case getting the bandwidth necessary for a 20 fs gate pulse is detrimental to getting a short blue pulse.

Crystal stuff:

How about calibrating the BBO crystal angle using the Hg lamp gated with the red and detecting with the CCD? Dither crystal angle to see how the spectrum changes and see the change in line intensities. Might want to filter the UV frequencies before mixing.

A thought about background counts:

It seems that waiting several minutes between exposing the CCD to bright light and collecting real data is a very good idea. There seems to be quite a bit of extra thermal noise that goes away with time.

Appendix F

Listing of the C version of the Phase Retrieval Algorithm

```
// Main program calls

#include <stdio.h>
#include <stdlib.h>
#include <CoreServices/CoreServices.h>
#include "main.h"
#include <gsl/gsl_min.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_linalg.h>
#include <time.h>
#include <string.h>


static UnsignedWide startTime, endTime;

void StartClock ( void ) {
      Microseconds( &startTime );
}


void StopClock( float *call_time ) {
      Microseconds( &endTime );

      if ( endTime.hi == startTime.hi )
            *call_time = ( float ) ( endTime.lo - startTime.lo );
      else
            *call_time = -1.0f;
}


FILE *traceDfile;
//FILE *gateDfile;
FILE *prevDfile;
FILE *ofile;
FILE *gfile;
FILE *zfile;
FILE *paramfile;
FILE *outparamfile;
FILE *progfile;

int main () {
```

```
UInt32        n=16;
UInt32        nsq=n*n;
UInt32      log2n=log2(n);
double      scale;
DSPDoubleSplitComplex gate, fluor,stpoint;
DSPDoubleSplitComplex gate2N, sig, tmp, workSpaceCMPLX;
double      *sigTmp;
double      *workSpaceNA,*workSpaceNB;
double      *workSpaceNC, *zeros;
double      *workSpace2NA,*gateAbsSq;
double      *gradient;
double      *zStore;
double lowestZ=100.0;
DSPDoubleSplitComplex fluorTmp;
int ni,nj;
FFTSetupD      setup;
__CLPK_doublecomplex *C;
__CLPK_doublecomplex *uu;
__CLPK_doublecomplex *vt;
__CLPK_doublecomplex *workC;
__CLPK_doublereal *workR;
__CLPK_integer lwork;
__CLPK_doublereal * svals;
char JOBU='A';
char JOBVT = 'N';
__CLPK_integer INFO;
//char paramName[]="xfrog_pfile.txt";
//char prevRetrievName[]="out.txt";
int min,maxh,prev,storeProg,everyith;
char junk;
CBLAS_INDEX indMax;
double mul=34;
int gothead=0;
char infoheader[100];
char infoheader2[100];
char infoheader3[100];
//char * hstflag="{";
char hstflag[1]="{";
int svd=1, rev=0, verb=1, oc=1, expon;
float overb=1.1, bs;
double minus1=-1.0;
//printf("%s\n",hstflag);
//printf("SVD? (1=yes,0=no)  ");
//scanf("%1d",&svd);
svd=0;
printf("Reverse gate? (1=yes,0=no)  ");
scanf("%1d",&rev);
printf("Print stats to screen? (1=yes,0=no)  ");
scanf("%1d",&verb);

//printf("Overcorrection? (1=yes,0=no)  ");
//scanf("%1d",&oc);
```

```
oc=0;
if (oc==1)
      {
      printf("base ");
      scanf("%f",&bs);
      printf("exponent ");
      scanf("%d",&expon);
      printf("pow(%f,%d)\n",bs, expon);
      }
paramfile=fopen("xfrog_pfile.txt","r");
//junk='0';
//look for header
fscanf(paramfile,"%c", &junk);
if (strcmp(&junk,hstflag)==0)
      {
      printf("header found\n");
      fscanf(paramfile,"%[^}]",infoheader);
      fscanf(paramfile,"%c", &junk);
      gothead=1;
      }
      else
      {
      rewind(paramfile);
      }

fscanf(paramfile,"%d",&n);
printf("n=%d\n",n);
nsq=n*n;
log2n=log2(n);
double *dData;
dData = ( double* ) malloc ( n * n * sizeof ( double ) );

//use a previous result to start
fscanf(paramfile,"%d",&prev);
//printf("prev=%d\n",prev);
//Use vanilla (0) or pcgp (1)
fscanf(paramfile,"%d",&min);
//printf("minimizer?%d\n",min);
//Number of iterations;
fscanf(paramfile,"%d",&maxh);
//printf("iterations=%d\n",maxh);
//store every ith fluorescence field? (1=yes, 0=no)
fscanf(paramfile,"%d",&storeProg);
//printf("store progress?%d\n",storeProg);
//store how often?
fscanf(paramfile,"%d",&everyith);
//printf("everyith=%d\n",everyith);

//load experimental data
for(nj=0;nj<n;nj+=1)
      {
      for(ni=0;ni<n;ni+=1)
```

```
                    {
                    fscanf(paramfile,"%lf",&dData[nj*n+ni]);
//fscanf(paramfile,"%lf",&dData[ni*n+nj]);
                    dData[nj*n+ni]=sqrt(dData[nj*n+ni]);
//dData[ni*n+nj]=sqrt(dData[ni*n+nj]);

                    }
               }

      printf("Start from previous result stored in out.txt?
(1=yes,0=no) ");
      scanf("%d",&prev);

      printf("iterations ");
      scanf("%d",&maxh);
      printf("Store results every ith iteration ");
      scanf("%d",&everyith);

      printf("\n");
      //printf("data2=%6.5f\n",dData[32000]);
      //allocate mem for all of the big arrays
      tmp.realp = ( double* ) malloc ( n * sizeof ( double ) );
      tmp.imagp = ( double* ) malloc ( n * sizeof ( double ) );
      gate.realp = ( double* ) malloc ( n * sizeof ( double ) );
      gate.imagp = ( double* ) malloc ( n * sizeof ( double ) );
      fluor.realp = ( double* ) malloc ( n *  sizeof ( double ) );
      fluor.imagp = ( double* ) malloc ( n *  sizeof ( double ) );
      workSpaceNA = ( double* ) malloc ( n * n * sizeof ( double ) );
      //looks like a lot of work space, but even for n=2^14 the total
is less than 4MB
      workSpaceNB = ( double* ) malloc ( n * n * sizeof ( double ) );
      workSpaceNC = ( double* ) malloc ( n * n * sizeof ( double ) );
      workSpace2NA = ( double* ) malloc (2 * n * sizeof ( double ) );
      gateAbsSq = ( double* ) malloc (2 * n * sizeof ( double ) );
      gate2N.realp = ( double* ) malloc (2 * n * sizeof ( double ) );
      gate2N.imagp = ( double* ) malloc (2 * n * sizeof ( double ) );
      zeros = ( double* ) malloc ( n * sizeof ( double ) );
      sig.realp = ( double* ) malloc ( n * n * sizeof ( double ) );
      sig.imagp = ( double* ) malloc ( n * n * sizeof ( double ) );
      sigTmp = ( double* ) malloc ( n * n * sizeof ( double ) );
      gradient = ( double* ) malloc ( n * sizeof ( double ) );
      fluorTmp.realp = ( double* ) malloc ( n * sizeof ( double ) );
      fluorTmp.imagp = ( double* ) malloc ( n * sizeof ( double ) );
      zStore=( double*) malloc (maxh * sizeof ( double ) );
      C = (__CLPK_doublecomplex*) malloc (2 * n * n * sizeof (double)
);
      uu = (__CLPK_doublecomplex*) malloc (2 * n * n * sizeof (double)
);
      vt = (__CLPK_doublecomplex*) malloc (2 * n * n * sizeof (double)
);
      svals = ( __CLPK_doublereal* ) malloc ( n * sizeof ( double ) );
```

```
      workC =  (__CLPK_doublecomplex*) malloc (n * n * sizeof ( double
));//10 * n * sizeof (double) );
      workR = ( __CLPK_doublereal* ) malloc ( n * n * sizeof ( double )
);
      lwork=n*n;
      __CLPK_integer nclapack=n;

      if (prev==1)
            {
            prevDfile=fopen("out.txt","r");
            if(prevDfile==NULL)
                  {
                  printf("Bad file name Beeauch!\n");
                  }
            //look for one line header
            fscanf(prevDfile,"%c", &junk);
            if (strcmp(&junk,hstflag)==0)
                  {
                  printf("header found in previous retrieval\n");
                  fscanf(prevDfile,"%[^}]",&infoheader2);
                  fscanf(prevDfile,"%c", &junk);
                  gothead=1;

                  //look for second line of header if it exists
                  fscanf(prevDfile,"%c", &junk);
                  fscanf(prevDfile,"%c", &junk);
                  if (strcmp(&junk,hstflag)==0)
                        {
                        fscanf(prevDfile,"%[^}]",&infoheader3);
                        fscanf(prevDfile,"%c", &junk);
                        }
                  else
                        {
                        rewind(prevDfile);
                        fscanf(prevDfile,"%c", &junk);
                        fscanf(prevDfile,"%[^}]",&infoheader3);
                        fscanf(prevDfile,"%c", &junk);
                        }

                  }
            else rewind(prevDfile);


            for (ni=0; ni<n; ni+=1)
                  {
                  fscanf(prevDfile,"%le",&fluor.realp[ni]);
                  fscanf(prevDfile,"%le",&fluor.imagp[ni]);
                  //printf("here%f\n",fluor.realp[ni]);
                  }

            fclose(prevDfile);
            }
```

```
      if (rev==0)
            {
            for (ni=0; ni<n; ni+=1)
                  {
                  fscanf(paramfile,"%le",&gate.realp[ni]);
                  fscanf(paramfile,"%le",&gate.imagp[ni]);
                  }
            }
            else
            {
                  for (ni=n-1; ni>=0; ni-=1)
                        {
                        fscanf(paramfile,"%le",&gate.realp[ni]);
                        fscanf(paramfile,"%le",&gate.imagp[ni]);
                        }
            }
      //use a random guess for the initial guess if no previous exists
double re, im;
time_t Tm;
char * ctm;
time(&Tm);
      if (prev==0)
            {
            ctm=ctime(&Tm);
            printf("%-26s\n",ctm);
            srandom(time(&Tm));
            for(ni=0;ni<n;ni+=1)
                  {
                  if(ni<n/4) fluor.realp[ni]=1;//0.01;
                  else fluor.realp[ni]=1.0;
                  //fluor.imagp[ni]= exp( -1.0*pow((ni-n/2.0-
1.0)/10.0,2.0) );
                  //srand(121295);

                  //fluor.imagp[ni]=random();
                  fluor.imagp[ni]=1;
                  }
            //printf("fluor[%3d] = (%4.3e,%4.3e)\n",
n/2,fluor.realp[n/2],fluor.imagp[n/2]);
            //indMax=cblas_idamax(n,fluor.realp,1);
            //mul=1/fluor.realp[indMax];
            //vsmulD(fluor.realp,1,&mul,fluor.realp,1,n);
            //indMax=cblas_idamax(n,fluor.imagp,1);
            //printf("max=%5.4e\n",fluor.imagp[indMax]);

            //mul=6.283185/fluor.imagp[indMax];
            //vsmulD(fluor.imagp,1,&mul,fluor.imagp,1,n);
            /*for(ni=0;ni<n;ni+=1)
                  {
                  re=fluor.realp[ni] * cos(fluor.imagp[ni]-3.141593);

                  im=fluor.realp[ni] * sin(fluor.imagp[ni]-3.141593);
```

```
                fluor.realp[ni]=re;
                fluor.imagp[ni]=im;

      //printf("(%4.3e,%4.3e)\n",fluor.realp[ni],fluor.imagp[ni]);
                }
            indMax=cblas_idamax(n,fluor.realp,1);
            mul=1/fluor.realp[indMax];
            vsmulD(fluor.realp,1,&mul,fluor.realp,1,n);
            indMax=cblas_idamax(n,fluor.imagp,1);
            mul=1/fluor.imagp[indMax];
            vsmulD(fluor.imagp,1,&mul,fluor.imagp,1,n);*/

            //printf("fluor[%3d] = (%4.3e,%4.3e)\n",
n/2,fluor.realp[n/2],fluor.imagp[n/2]);
            }

      /* zmmulD gives the outer product of gate and fluor. With gate
first and fluor second, the resulting matrix looks like
      f0g0        f0g1        f0g2        ...
      f1g0        f1g1        f1g2        ...
      f2g0        f2g1        f2g2        ...

      if the array indices are ploted
      [0]         [n]         [2n]        ...   [(n-1)*n]
      [1]         [n+1] [2n+1]      ...
      [2]         [n+2] [2n+2]      ...
      [3]         [n+3] [2n+3]      ...
      ...
      [n-1]       [2n-1]      [3n-1]      ...   [n*n-1]

      Using this construction, the columns are constant in tau(delay
time) and the rows are constant in t.
      This is the same convention as in Trebino's FROG book.
      */

/*    zmmulD(&gate,1,&fluor,1,&sig,1,n,n,1);

      rowTater2L(n, sig.realp);
      rowTater2L(n, sig.imagp);
      columnArranger2(n, sig.realp);
      columnArranger2(n, sig.realp);
      rowTater2R(n, sig.realp);*/

      setup = create_fftsetupD ( log2n, FFT_RADIX2 );
      if ( setup == NULL )
            {
            printf ( "\nFFT_Setup failed to allocate enough memory.\n"
);
            exit ( 0 );
            }
      scale = ( double ) 1.0 / n;
```

```
/////set up copies of gate for the gradient calculation
      double minustwo, two;
      minustwo=-2.0;
      two=2.0;
      for(ni=0;ni<(n+n);ni++)
            {
            gate2N.realp[ni]=0;
            gate2N.imagp[ni]=0;
            workSpace2NA[ni]=0;
            gateAbsSq[ni]=0;
            }

      for(ni=0;ni<n;ni++)
            {
            zeros[ni]=0.0;
            }

      vsmulD(gate.realp,1,&minustwo,&gate2N.realp[n/2],1,n);
      vsmulD(gate.imagp,1,&two,&gate2N.imagp[n/2],1,n);
      vsqD(gate.realp,1,&gateAbsSq[n/2],1,n);
      vsqD(gate.realp,1,&workSpace2NA[n/2],1,n);
      vaddD(gateAbsSq,1,workSpace2NA,1,gateAbsSq,1,2*n);
      vsmulD(gateAbsSq,1,&two,gateAbsSq,1,2*n);
      double bill;
      int byE, byColumn, mfft;
      byE=0;
      byColumn=0;
      mfft=0;
      double mag;
      float time;
      int h;

      ////////////////////////////////////////////////////1-D minimizer
variables and setup
      int status;
      const gsl_min_fminimizer_type *T;
      gsl_min_fminimizer *s;
      int iter = 0, max_iter = 5000;
      double m=10,mg=2;
      double a =-10000, b = 10000,ag =0.001, bg = 100;
      gsl_function Z,G;

      struct z_params params={      sig, fluor,
                                    gate,  gradient,
                                    fluorTmp, n,
                                    workSpaceNA, workSpaceNB,
                                    workSpaceNC, workSpace2NA,
                                    zeros};
      struct g_params paramsG={sig,  dData,
                                    workSpaceNA, workSpaceNB,
                                    workSpaceNC,
                                    zeros, n };
```

```
	Z.function=&zfor_min2;
	Z.params=&params;
	G.function=&gfor_min;
	G.params=&paramsG;

	T = gsl_min_fminimizer_brent;

	double realMax, imagMax;


			a =-10000;
			b = 10000;
			m=10;

	progfile=fopen("prog.txt","w"); //open file to periodically save
the output of fluor
	if (gothead==1) fprintf(progfile,"{%s}\n",infoheader);
	StartClock ( );
	for(h=0;h<maxh;h++)	//if (1)//mfft)
	////////Main loop start ////////////////////
	{

		for (ni=0; ni<n; ni++)
			{

	vsmulD(fluor.realp,1,&gate.realp[ni],workSpaceNA,1,n);

	vsmulD(fluor.imagp,1,&gate.imagp[ni],workSpaceNB,1,n);

	vsubD(workSpaceNA,1,workSpaceNB,1,&sig.realp[ni*n],1,n);

	vsmulD(fluor.realp,1,&gate.imagp[ni],workSpaceNA,1,n);

	vsmulD(fluor.imagp,1,&gate.realp[ni],workSpaceNB,1,n);

	vaddD(workSpaceNA,1,workSpaceNB,1,&sig.imagp[ni*n],1,n);
			}
		rowTater2L(n, sig.realp);
		rowTater2L(n, sig.imagp);
		columnArranger2(n, sig.realp);
		columnArranger2(n, sig.imagp);
		if (oc==1 && h>30) overb=pow(bs,1.0+h/expon);
		else overb=1;

	//	printf("overb=%7.6f\n",overb);
		if (1)
		{
		for(ni=0;ni<nsq;ni+=n)
			{

	memcpy(fluorTmp.realp,&sig.realp[ni],(n*sizeof(double)));
```

```
      //load real part of one column of the calculated signal into
fluorTmp

      memcpy(fluorTmp.imagp,&sig.imagp[ni],(n*sizeof(double))); //load
imag part of one column of the calculated signal into fluorTmp
                  fft_zipD(setup,&fluorTmp,1,log2n,FFT_FORWARD);   //do
the FFT on the current column

      memcpy(&sig.realp[ni],fluorTmp.realp,(n*sizeof(double)));

      memcpy(&sig.imagp[ni],fluorTmp.imagp,(n*sizeof(double)));
                  }

            if (1)
            {
            iter=0;
            ag =-1000.0;
            bg = 1000.0;
            //if (h==1) mg=1.0;
            if (h==0) mg=1;
            //mg=1.01;
            s = gsl_min_fminimizer_alloc (T);
            gsl_min_fminimizer_set(s,&G, mg, ag, bg);
            do
            {
                  iter++;
                  status = gsl_min_fminimizer_iterate (s);
                  mg = gsl_min_fminimizer_x_minimum (s);
                  ag = gsl_min_fminimizer_x_lower (s);
                  bg = gsl_min_fminimizer_x_upper (s);
                  status = gsl_min_test_interval (ag, bg, 0.00001,
0.0);
            }

            while (status == GSL_CONTINUE && iter < max_iter);

            if (verb==1) printf("%4d  [%d]  G=%6.5e  minimum at
x=%6.5f\n",h,iter,GSL_FN_EVAL(&G,mg),mg);
            zStore[h]=  GSL_FN_EVAL(&G,mg);
            gsl_min_fminimizer_free(s);
            }
            //if (oc==0)
                  //{
                  vDSP_zvabsD(&sig,1,workSpaceNA,1,n*n);
                  workSpaceCMPLX.realp=workSpaceNB;
                  workSpaceCMPLX.imagp=workSpaceNC;

      vDSP_zrvdivD(&sig,1,workSpaceNA,1,&workSpaceCMPLX,1,n*n);
                  vDSP_zrvmulD(&workSpaceCMPLX,1,dData,1,&sig,1,n*n);
                  //printf("\nsig=%.5f",sig.realp[1000]);
                  //printf("\nsig=%.5f\n",sig.imagp[1000]);
```

```
                        //}
                  /*else
                        {
                        vDSP_zvabsD(&sig,1,workSpaceNA,1,n*n);
                        vDSP_vdivD(dData,1,workSpaceNA,1,workSpaceNB,1,n*n);
                        for (nj=0; nj<(n*n);nj++)
                              {

      workSpaceNA[nj]=workSpaceNB[nj];//pow(workSpaceNB[nj],(double)ove
rb);
                              }
                              printf("\nsig=%.5f",sig.realp[1000]);
                              printf("\noverb=%.5f",overb);
                        workSpaceCMPLX.realp=workSpaceNB;
                        workSpaceCMPLX.imagp=workSpaceNC;
                        printf("\nwork=%.5f",workSpaceCMPLX.realp[1000]);

      memcpy(workSpaceCMPLX.realp,sig.realp,(n*n*sizeof(double)));
                        printf("\nwork=%.5f",workSpaceCMPLX.realp[1000]);

      memcpy(workSpaceCMPLX.imagp,sig.imagp,(n*n*sizeof(double)));

      vDSP_zrvmulD(&workSpaceCMPLX,1,workSpaceNA,1,&sig,1,n*n);
                        printf("\nsig=%.5f",sig.realp[1000]);
                        printf("\nsig=%.5f\n",sig.imagp[1000]);
                        }*/
                  }

                  for(ni=0;ni<nsq;ni+=n)
                        {


      memcpy(fluorTmp.realp,&sig.realp[ni],(n*sizeof(double)));

      memcpy(fluorTmp.imagp,&sig.imagp[ni],(n*sizeof(double)));
                        fft_zipD(setup,&fluorTmp,1,log2n,FFT_INVERSE);


      memcpy(&sig.realp[ni],fluorTmp.realp,(n*sizeof(double)));

      memcpy(&sig.imagp[ni],fluorTmp.imagp,(n*sizeof(double)));
                        }


                  vsmulD( sig.realp, 1, &scale, sig.realp, 1, nsq);
                  vsmulD( sig.imagp, 1, &scale, sig.imagp, 1, nsq);

                  columnArranger2(n, sig.realp);
                  columnArranger2(n, sig.imagp);
                  rowTater2R(n, sig.realp);
                  rowTater2R(n, sig.imagp);
```

```
            ////////////////////////////PCGP/////////////////////
            if(svd==0)
                  {
                  zmmulD(&sig,1,&fluor,1,&fluorTmp,1,n,1,n);
                  mtransD(sig.realp,1,sigTmp,1,n,n);
                  memcpy(sig.realp,sigTmp, (nsq*sizeof(double)));
                  mtransD(sig.imagp,1,sigTmp,1,n,n);
                  vDSP_vsmulD(sigTmp,1,&minus1,sig.imagp,1,n*n);
                  zmmulD(&sig,1,&fluorTmp,1,&fluor,1,n,1,n);
                  }
            else
                  {
                  ztocD(&sig,1,(DSPDoubleComplex*)C,2, n*n);

                  if (0)
                        {fprintf(progfile," Cre \t Cim \n");
                        for (ni=0; ni<n*n; ni+=1)
                              {
                              fprintf(progfile,"%10.9e\t",C[ni].r);
                              fprintf(progfile,"%10.9e\n",C[ni].i);
                              }
                        }


                  zgesvd_( &JOBU, &JOBVT, &nclapack, &nclapack, C,
&nclapack, svals, uu, &nclapack, vt, &nclapack, workC, &lwork, workR,
&INFO);
                  indMax=cblas_idamax(n,svals,1);
                  ctozD((DOUBLE_COMPLEX*) uu, 2, &fluor,1, n);
                  }
            vDSP_zvabsD(&fluor,1,workSpaceNA,1,n);
            indMax=cblas_idamax(n, workSpaceNA, 1);
            //indMax=cblas_idamax(n, fluor.realp, 1);
            mul=1/workSpaceNA[indMax];
            vsmulD(fluor.realp,1,&mul,fluor.realp,1,n);
            vsmulD(fluor.imagp,1,&mul,fluor.imagp,1,n);
            //printf("\n
max[%i]=%4.3f\t",indMax,sqrt(pow(fluor.realp[indMax],2)+pow(fluor.imagp
[indMax],2)));

            if (0)
                  {fprintf(progfile,"\n uure \t uuim \n");
                  for (ni=0; ni<n*n; ni+=1)
                        {
                        //fprintf(progfile,"%d\t",h);
                        fprintf(progfile,"%10.9e\t",uu[ni].r);
                        fprintf(progfile,"%10.9e\n",uu[ni].i);
                        }

                  }

            if (zStore[h] <= lowestZ)
```

```
                    {
                    lowestZ=zStore[h];
                    gfile=fopen("lowest.txt","w");
                    fprintf(gfile,"error[%d]=%.14f\r",h,zStore[h]);
                    for (ni=0; ni<n; ni+=1)
                          {
                          fprintf(gfile,"%.14f\t",fluor.realp[ni]);
                          fprintf(gfile,"%.14f\n",fluor.imagp[ni]);
                          }

                    fclose(gfile);
                    }
              if (storeProg==1)
                    {
                    if (h % everyith==0)
                          {fprintf(progfile,"iteration\tRe\tIm\n");
                          for (ni=0; ni<n; ni+=1)
                                {
                                fprintf(progfile,"%d\t",h);

      fprintf(progfile,"%10.9e\t",fluor.realp[ni]);

      fprintf(progfile,"%10.9e\n",fluor.imagp[ni]);
                                }

                          }

                    }
      }
      StopClock ( &time );
      //printf("\n info=%d\n",INFO);
      /*gfile=fopen("grade.txt","w"); //open file for output of
gradient
      if (gothead==1) fprintf(gfile,"{%s}\n",infoheader);
      for (ni=0; ni<n; ni+=1)
      {
            fprintf(gfile,"%10.9e\n",gradient[ni]);
      }
      fclose(gfile);*/

      //open and write to output files
      ofile=fopen("out.txt","w"); //open file for output of fluor
      //if (gothead==1) fprintf(ofile,"{%s}\n",infoheader);
      //fprintf(ofile, "{Grid points=%d Iterations=%d
",n,maxh);//retrieval params for the header
      //if (min==0) fprintf(ofile,"Vanilla method");//retrieval params
for the header
      //else fprintf(ofile,"PCGP method");//retrieval params for the
header
      //if (prev==1) fprintf(ofile,"started from previous result from
out.txt");//retrieval params for the header
      //fprintf(ofile,"}\n");
```

```
      for (ni=0; ni<n; ni+=1)
            {
            fprintf(ofile,"%.14f\t",fluor.realp[ni]);
            fprintf(ofile,"%.14f\n",fluor.imagp[ni]);
            }
      fclose(ofile);
      fclose(progfile);
      zfile=fopen("z.txt","w"); //open file to save the error function
z
      if (gothead==1) fprintf(zfile,"{%s}\n",infoheader);
      fprintf(zfile, "{Grid points=%d Iterations=%d ",n,maxh);
//retrieval params for the header
      if (min==0) fprintf(zfile,"Vanilla method");//retrieval params
for the header
      else fprintf(zfile,"PCGP method");
      if (prev==1) fprintf(zfile," started from previous result from
out.txt");//retrieval params for the header
      fprintf(zfile,"}\n");
      for (ni=0; ni<maxh; ni+=1)
            {

            fprintf(zfile,"%.14f\n",zStore[ni]);
            }
      time/=1000000;
      fprintf(zfile,"\nTime=%10.0f secs for %i
iterations\n\n",time,maxh);//retrieval params for the footer
      fclose(zfile);


      destroy_fftsetupD ( setup );
      free(gate.realp);
      free(gate.imagp);
      free(fluor.realp);
      free(fluor.imagp);
      free(workSpaceNA);
      free(workSpaceNB);
      free(workSpaceNC);
      free(workSpace2NA);
      free(gateAbsSq);
      free(gate2N.realp);
      free(gate2N.imagp);
      free(zeros);
      free(sig.realp);
      free(sig.imagp);
      free(sigTmp);
      free(gradient);
      free(fluorTmp.realp);
      free(fluorTmp.imagp);
      free(dData);
      free(zStore);
      free(C);
      free(uu);
```

```
      free(vt);
      free(svals);
      free(workC);
      free(workR);
    return 0;
}


// functions for the gsl minimizer

/*
 *  funcsformin.c
 *  xfrog
 *
 *  Created by Benjamin Langdon on 4/26/05.
 *  Copyright 2005 __MyCompanyName__. All rights reserved.
 *
 */


#include <gsl/gsl_min.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_math.h>
#include <stdio.h>
#include <stdlib.h>
#include <CoreServices/CoreServices.h>
#include "main.h"
double gfor_min (double x, void * p)
{
      struct g_params * params = (struct g_params *) p;
      double * dData= (params->dData);
      DOUBLE_COMPLEX_SPLIT sig= (params->sig);
      int n = (params->n);
      int ni;
      double G=0.0;
      int nsq=n*n;
      for (ni=0; ni<nsq; ni++) G+=pow(pow(dData[ni],2) - x *
(pow(sig.realp[ni],2) + pow(sig.imagp[ni],2)),2);
      //G+=workSpaceNC[ni];

      G=sqrt(G)/nsq;
      return G;
}

double zfor_min2 (double x, void * p)
{
      struct z_params * params = (struct z_params *) p;
      DOUBLE_COMPLEX_SPLIT sig= (params->sig);
      DOUBLE_COMPLEX_SPLIT fluor=(params->fluor);
      DOUBLE_COMPLEX_SPLIT gate=(params->gate);
      double * gradient=(params->gradient);
      DOUBLE_COMPLEX_SPLIT  fluorTmp=(params->fluorTmp);
```

```
        int n = (params->n);
        int ni;
        double *workSpaceNA=(params->workSpaceNA);
        double *workSpaceNB=(params->workSpaceNB);
        double *workSpaceNC=(params->workSpaceNC);
        double *workSpace2NA=(params->workSpace2NA);
        double *zeros=(params->zeros);
        double Z=0.0;
        DOUBLE_COMPLEX_SPLIT Tmp;
//      Tmp.realp = ( double* ) malloc ( n *  sizeof ( double ) );
//      Tmp.imagp = ( double* ) malloc ( n *  sizeof ( double ) );

//if input sig is in "outer product" form then there is no need for
//any rearrangement before comparison
        memcpy (workSpace2NA,zeros, (n*sizeof(double)));
        vsmulD(gradient,1,&x,fluorTmp.realp,1,n);
        vsmulD(gradient,1,&x,fluorTmp.imagp,1,n);
        zvmulD(&fluor,1,&fluorTmp,1,&fluorTmp,1,n,1);
        for (ni=0; ni<n; ni++)
              {
              vsmulD(fluorTmp.realp,1,&gate.realp[ni],workSpaceNA,1,n);
              vsmulD(fluorTmp.imagp,1,&gate.imagp[ni],workSpaceNB,1,n);
              vsubD(workSpaceNA,1,workSpaceNB,1,workSpaceNC,1,n); //C has
real part
              vsmulD(fluorTmp.realp,1,&gate.imagp[ni],workSpaceNA,1,n);
              vsmulD(fluorTmp.imagp,1,&gate.realp[ni],workSpaceNB,1,n);
              vaddD(workSpaceNA,1,workSpaceNB,1,workSpaceNB,1,n); //B has
imag part

              Tmp.realp=workSpaceNC;
              Tmp.imagp=workSpaceNB;
              //memcpy(Tmp.realp,workSpaceNC,(n*sizeof(double)));
        //new signal(k+1)
              //memcpy(Tmp.imagp,workSpaceNB,(n*sizeof(double)));

              vsubD(&sig.realp[ni],1,Tmp.realp,1,Tmp.realp,1,n);
              vsubD(&sig.imagp[ni],1,Tmp.imagp,1,Tmp.imagp,1,n);

              vsqD(Tmp.realp,1,Tmp.realp,1,n);
              vsqD(Tmp.imagp,1,Tmp.imagp,1,n);
              vaddD(Tmp.realp,1,Tmp.imagp,1,Tmp.imagp,1,n);
              vaddD(Tmp.imagp,1,workSpace2NA,1,workSpace2NA,1,n);
              }
        for (ni=0; ni<n; ni++)   Z+=workSpace2NA[ni];
        //free(Tmp.realp);
        //free(Tmp.imagp);
        return Z;
}

//"rowtater"

/*
```

```
 *  rowTater.c
 *  testing
 *
 *  Created by Benjamin Langdon on 4/5/05.
 *  Copyright 2005. All rights reserved.
 *
 */

#include "rowTater.h"


////////////////////////////////////////////////////////////
//                                                        //
//          Row Rotator--Left                             //
//          To be used _BEFORE_ forward fft               //
//          Takes an inline stored nxn matrix and rotates //
//          each row left by the number of the row.       //
//                                                        //
//          Before                  after rotation        //
//                                                        //
//    11    12    13    14          11    12    13    14  //
//    21    22    23    24          22    23    24    21  //
//    31    32    33    34          33    34    31    32  //
//    41    42    43    44          44    41    42    43  //
////////////////////////////////////////////////////////////


//ni is the row number (0->N-1), nj is column number (0->N-1)
void rowTaterL(int n, double *F){
      int ni, nj;
      double *G;
      G = ( double* ) malloc ( n * n * sizeof ( double ) );
      //printf("n=%d\tF[16]=%4.2f\n",n,F[16]);
      for(ni=0; ni<n; ni+=1)
            {
            for( nj=0; nj<(n-ni); nj+=1)
                  {
                  G[ni*n+nj]=F[(nj+ni)+(n*ni)];
                  }
                  //printf("\n");

            for( nj=(n-ni); nj<n; nj+=1)
                  {
                  G[ni*n+nj]=F[(nj-n+ni)+(n*ni)]; //G[ni*n+nj]=F[(n-nj-
1)+(n*ni)]; (reverses order)
                  }
                  //printf("\n");

            }
            memcpy ( F, G, (n* n * sizeof ( double ) ) );
            free(G);
            return;
```

```
}

void columnArranger(int n, double *F){
      int ni, nj;
      double *G;
      G = ( double* ) malloc ( n * n * sizeof ( double ) );

////////////////////////////////////////////////////////////
//                                                        //
//          Column Re-arranger                            //
//          To be used before and after fft               //
//          Takes an inline stored nxn matrix and shifts  //
//          each column right by n/2-1 then reverses their   //
//          order. Two applications of this returns the   //
//          original.                                     //
//                                                        //
//          Before                  after rotation        //
//                                                        //
//    11    12    13    14          12    11    14    13  //
//    22    23    24    21          23    22    21    24  //
//    33    34    31    32          34    33    32    31  //
//    44    41    42    43          41    44    43    42  //
////////////////////////////////////////////////////////////

      for(ni=0; ni<n; ni+=1)
            {
            for (nj=0; nj<n/2 ; nj+=1)
                  {
                        G[ni*n+nj]=F[-nj+n/2-1+(n*ni)];
                  }
            for (nj=n/2; nj<n ; nj+=1)
                  {
                        G[ni*n+nj]=F[-nj+3*n/2-1+(n*ni)];
                  }
            }
      memcpy ( F, G, (n* n * sizeof ( double ) ) );
      free(G);
      return;
}


////////////////////////////////////////////////////////////
//                                                        //
//          Row Rotator--Right                            //
//          To be used _AFTER_ reverse fft                //
//          Takes an inline stored nxn matrix and rotates    //
//          each row left by the number of the row.       //
//                                                        //
//          Before                  after rotation        //
//                                                        //
//    11    12    13    14          11    12    13    14  //
```

```
//    22    23    24    21          21    22    23    24          //
//    33    34    31    32          31    32    33    34          //
//    44    41    42    43          41    42    43    44          //
////////////////////////////////////////////////////////////////////


void rowTaterR(int n, double *F){
      int ni, nj;
      double *G;
      G = ( double* ) malloc ( n * n * sizeof ( double ) );
      //printf("n=%d\tF[16]=%4.2f\n",n,F[16]);

      for(ni=0; ni<n; ni+=1)
            {
            for( nj=0; nj<(n-ni); nj+=1)
                  {
                  G[ni*(n+1)+nj]=F[ni*n+nj];
                  }
            for( nj=(n-ni); nj<n; nj+=1)
                  {
                  G[ni*n+(nj-n+ni)]=F[ni*n+nj];
                  }
            }
      memcpy ( F, G, (n* n * sizeof ( double ) ) );
      free(G);
      return;
}

////////////////////////////////////////////////////////////////////
//                                                                  //
//          Row Rotator2--Left                                      //
//          To be used _BEFORE_ forward fft                  //
//          Takes an inline stored nxn matrix and rotates    //
//          each row left by the number of the row.          //
//                                                                  //
//          Before                   after rotation                 //
//                                                                  //
//    11    12    13    14          11    12    13    14          //
//    21    22    23    24          22    23    24    21          //
//    31    32    33    34          33    34    31    32          //
//    41    42    43    44          44    41    42    43          //
////////////////////////////////////////////////////////////////////


//ni is the row number (0->N-1), nj is column number (0->N-1)
void rowTater2L(int n, double *F){
      int ni, nj;
      double *G;
      G = ( double* ) malloc ( n * n * sizeof ( double ) );
      //printf("n=%d\tF[16]=%4.2f\n",n,F[16]);
      for(nj=0; nj<n; nj+=1) //row iterater
            {
```

```
            for( ni=0; ni<(n-nj); ni+=1)//column iterater
                  {
                  G[nj+ni*n]=F[nj+n*ni+n*nj];
                  //printf("%i\t",(nj+n*ni+n*nj));
                  }
                  //printf("\n");

            for( ni=(n-nj); ni<n; ni+=1)
                  {
                  G[nj+ni*n]=F[nj+n*ni-n*(n-nj)]; //G[ni*n+nj]=F[(n-nj-
1)+(n*ni)]; (reverses order)

                  }
                  //printf("\n");

            }
            memcpy ( F, G, (n* n * sizeof ( double ) ) );
            free(G);
            return;

}

void columnArranger2(int n, double *F){
      int ni, nj;
      double *G;
      G = ( double* ) malloc ( n * n * sizeof ( double ) );

////////////////////////////////////////////////////////////
//                                                        //
//          Column Re-arranger                            //
//          To be used before and after fft               //
//          Takes an inline stored nxn matrix and shifts  //
//          each column right by n/2-1 then reverses their   //
//          order. Two applications of this returns the   //
//          original.                                     //
//                                                        //
//          Before                  after rotation        //
//                                                        //
//    11    12    13    14          12    11    14    13  //
//    22    23    24    21          23    22    21    24  //
//    33    34    31    32          34    33    32    31  //
//    44    41    42    43          41    44    43    42  //
////////////////////////////////////////////////////////////

      for(nj=0; nj<n; nj+=1)
            {
            for (ni=0; ni<n/2 ; ni+=1)
                  {
                        G[ni*n+nj]=F[nj+(n/2-1-ni)*n];
                  }
            for (ni=n/2; ni<n ; ni+=1)
                  {
```

```
                        G[ni*n+nj]=F[nj+(3*n/2-1-ni)*n];
                        //printf("[%i]\t",(nj+(3*n/2-1-ni)*n));
                  }
                  //printf("\n");
            }
      memcpy ( F, G, (n* n * sizeof ( double ) ) );
      free(G);
      return;
}
////////////////////////////////////////////////////////////
//                                                        //
//          Row Rotator--Right                            //
//          To be used _AFTER_ reverse fft                //
//          Takes an inline stored nxn matrix and rotates    //
//          each row left by the number of the row.       //
//                                                        //
//          Before                  after rotation        //
//                                                        //
//    11    12    13    14          11    12    13    14  //
//    22    23    24    21          21    22    23    24  //
//    33    34    31    32          31    32    33    34  //
//    44    41    42    43          41    42    43    44  //
////////////////////////////////////////////////////////////


void rowTater2R(int n, double *F){
      int ni, nj;
      double *G;
      G = ( double* ) malloc ( n * n * sizeof ( double ) );
      //printf("n=%d\tF[16]=%4.2f\n",n,F[16]);

      for(ni=0; ni<n; ni+=1)
            {
            for( nj=0; nj<(n-ni); nj+=1)
                  {
                  G[nj+n*ni+n*nj]=F[ni*n+nj];
                  }
            for( nj=(n-ni); nj<n; nj+=1)
                  {
                  G[nj+n*ni-n*(n-nj)]=F[ni*n+nj];
                  }
            }
      memcpy ( F, G, (n* n * sizeof ( double ) ) );
      free(G);
      return;
}

// Data loader

/*
 *  dataloader.c
 *  xfrog
```

```
 *
 *  Created by Benjamin Langdon on 3/29/05.
 *  Copyright 2005. All rights reserved.
 *
 */

//#include "dataloader.h"
#include <stdio.h>
#include <complex.h>
FILE *traceDfile;

void loader (char *traceName, int n,double *ldData) {
      //double frogD256[256][256];
      printf("sub ->%s\n\nhey\n%i\n",traceName,traceDfile);
      traceDfile=fopen(traceName,"r");
      if(traceDfile==NULL)
      {
            printf("Bad file name Beeauch!\n");
      }
      n=n*n;
      int ni;
      printf("here1\n");

      for (ni=0; ni<n; ni+=1)
      {
            fscanf(traceDfile,"%lf",ldData[ni]);
            printf("here%f\n",ldData[ni]);
            //ldData[ni]=sqrt(ldData[ni]);
      }

      fclose(traceDfile);
      //ldData=&frogD256;
      return;
      }
```

Appendix G

Intstructions for use of the C version of the Phase Retrieval Algorithm

Introduction

These are the instructions for use of the Phase Retrival Algorithm written in C and compliled for PPC G4 or G5 processors running Mac OS 10.3.9 or higher. The interface of the program is the Unix command line.

The program requires an input data file that contains the properly re-sampled experimental FROG and gate pulse data, the number of points in one dimension, and an initial guess for the electric field that you wish to retrieve. The Phase Retrieval Algorithm can also be initialized with a guess from a separate data file containing only the initial guess. This is convient when you wish to use the final retrieved field from a previous retrieval as the start for another retrieval. All of the input files must be in the same directory as the executable C program file. The exact file structure for the input and output files are described in the next appendix.

Operation of the program

Place the executable file called "xfrog" in a folder along with the input file called "xfrog_pfile.txt". If you wish to use the results from a previous retrieval as the starting point, then you must also place that file, called "out.txt", in the same folder.

To start the program type "./xfrog" at the Unix promp and press return. Here is an example of what this should look like:

[Bs-Computer:/Deployment G4] benzyl% ./xfrog

Next, the program will prompt with the following:

Reverse gate? (1=yes,0=no)

This gives you the option of performing the retrieval with the time reverse of the gate pulse electric field stored in xfrog_pfile.txt. This is usefull if you determine that gate pulse output by the GRENOUILLE or other FROG device is time reversed. Any second harmonic generation FROG measurement has a time direction ambiguity that can cause the retrieved pulse to be time reversed from the real pulse.

The next prompt askes if you want the retrieval statistics to be printed to the screen after each iteration of the algorithm.

Print stats to screen? (1=yes,0=no)

Immediately after answering this promp, the program will print a line of text "n= " with the number of points in one dimension of the input FROG trace and in the input gate pulse. This number will be some power of 2 such as 256, 512, etc. This output is merly a check and number of points cannot be changed at this point, because it was set when the input data file, xfrog_pfile.txt, was first written.

Next the program askes if you want to use the result from a previous retrieval as the starting guess for the electric field.

Start from previous result stored in out.txt? (1=yes,0=no)

The result must have the same number points as one dimension of the input FROG trace and must have the same number of points as the gate pulse. The name of the file must be "out.txt" and it mus reside in the same folder as the xfrog program.

It should be noted that all responsed to the above prompts should be either a "1" or "0". Responding with letters such as "y" or "n" or words such as "yes" or "no" at best cause the program to abort and at worst cause the program to continue to execute with unpredictable consequences.

The next prompt askes for the number of iterations.

iterations

This is were you tell the program how many iterations you wish it to perform. Usually several hundred to several thousand iterations are required for the program to settle on an electric field that satisfies both the data and mathematical constraints.

The last prompt askes how often to save intermediate electric field results from the program:

Store results every ith iteration

For instance if you respond to this prompt with 10 the program will save the electric field after every 10^{th} iteration. The intermediate results will be saved in a file called " prog.txt".

Typing "1" at the "print stats to screen" prompt will result in text output to the terminal window:

```
0 [29] G=7.36867e-04  minimum at x=27.41799
1 [27] G=7.27454e-04  minimum at x=27.24825
2 [23] G=7.23107e-04  minimum at x=27.15500
3 [31] G=7.19866e-04  minimum at x=27.01863
4 [21] G=7.17174e-04  minimum at x=26.76300
5 [20] G=7.14836e-04  minimum at x=26.47076
6 [31] G=7.12748e-04  minimum at x=26.15353
7 [20] G=7.10777e-04  minimum at x=25.83199
```

8 [20] G=7.08897e-04 minimum at x=25.50890
9 [26] G=7.07123e-04 minimum at x=25.42575

The first number is the iteration number of the xfrog program. The second number, in square brakets [ ], is the number of iterations executed by the linear least squares subroutine in order to scale the calculated FROG trace to the input FROG trace. The scaling factor found by the least squares routine is listed at the end of the output line as "minimum at x=". The scaling factor x will fluctuate as the program iterates. The number "G=" is the error. This is the RMS difference between the scaled/calculated FROG trace and the input FROG trace. G will decrease as the program is allowed to iterate. The decrease in G indicates that the calculated FROG trace increasing resembles the input FROG trace.

Appendix H

Structure of the XFROG program Data files

The "xfrog" program uses one or, optionally, two data files for parameter and experimental data input. The file "xfrog_pfile.txt" is generated by the "X-FROG Tools" control panel within Igor Pro. The first line of the data file contains the number of points in each dimension of the re-sampled XFROG data as well as the number of points in the measured gate pulse. The next five lines contain unused input parameters also generated by the control panel in Igor Pro. These parameters are, instead, passed to the program via user input text at runtime.

```
256         ← Number of points in one dimension of the re-sampled data
0           ← Unused
1           ← Unused
20          ← Unused
1           ← Unused
1           ← Unused
0.824976    ← First point of the XFROG data
0.832729
0.840475
0.848212
...
```

The next section of the input file contains the XFROG data. The data is written one point per line written in time slices. This means that one frequency resolved gated fluorescence spectrum is written with one point per line, then the spectrum from the next time step is written in the same manner.

The next section is the gate pulse. Because the gate pulse electric field is complex, two real values (representing the real and imaginary parts) must be written for each time point. The complex value is written with the real and imaginary parts on the same line separated by a tab.

At runtime, the user has the option to initialize the phase retrieval algorithm with an initial guess other than the one supplied by the "xfrog" program. This is stored in a data filed named exactly "out.txt" contained in the same directory as the input data file and the xfrog program. The "out.txt" file has no header or other parameter information, it only has the complex values for reach time point of the initial electric field. The complex values are written in the same format as the gate pulse electric field with the real and imaginary parts written on the same line with a tab separating them. Supplying an initial guess this way allows the user to use the results from a previous phase retrieval as the starting point for a new retrieval. It also allows the user to investigate the effects of different initial guesses on the results of a retrieval provided those initial guesses are written in the proper format.

Another file created by the program as it is executing is called "lowest.txt". This contains the retrieved electric field with the lowest error. After several hundred or thousand iterations, this is the file that contains the algorithm's best retrieved field. In other words, if you are confident that the algorithm has completed sufficient iterations to converge to a single "best fit" field, then this is where it will be stored. This will most likely be different than the file called "out.txt". The output of "lowest.txt" looks like this:

```
error[19]=0.00069660265957                ← iteration number and error
0.18058149718323   0.10598473448804       ← start of complex output field
0.08259285881489   -0.01127155767068
```

0.07505946904748 0.02345209584429
-0.04377980727937 -0.02756832842721
-0.07144566255934 0.01810655218511
...

The first line contains the text "error[ ]=". The number in the brackets indicates the iteration number that contained the lowest error. The number after the equal sign is the error, or the RMS difference between the calculated signal and the real data. On the next line the output complex temporal field starts. From here, the file has the same format as the "out.txt" file with the real and imaginary parts of the field separated by a tab.

The last file that is output by the program is called "prog.txt", where "prog" is short for progress. This file contains the output complex field stored at regular intervals of the execution of the xfrog program. This is determined by the user at runtime at the prompt that asks to save after "every ith iteration". For instance, if the user answered 10 at the prompt, then the output field would be saved from the 0^{th}, 10^{th}, 20^{th}, etc iterations. A sample output from a "prog.txt" file is shown below. The fields from several saved iterations are separated by the column labels "iteration Re Im".

This file is useful if there is some kind of power failure or other catastrophic event that prevents the program from complete execution and from creating the "out.txt" or the "lowest.txt". It also is useful to trace the path that the algorithm takes to converge on a output field.

iteration	Re	Im	← column labels
0	2.052278630e-01	1.076600197e-01	← iteration number, complex field
0	7.819634949e-02	-3.693625447e-02	
0	4.596813211e-02	-5.667192199e-03	
0	-2.803062198e-02	6.611482166e-03	
0	-5.893040742e-02	2.532634527e-02	
0	-2.551946675e-02	6.936396333e-02	

...

iteration	Re	Im	← field from a different iteration
10	1.967704077e-01	9.814701053e-02	
10	6.322187393e-02	-3.817146349e-02	
10	6.317489995e-02	1.381997864e-02	
10	-4.873238425e-02	-1.846891788e-02	
10	-7.791622277e-02	2.048416389e-02	
10	-3.685942291e-02	7.895839597e	