Femtosecond Transition-State Spectroscopy of Chemical Reactions

Thesis by

Marcos Dantus

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California
1991

(Defended December 7, 1990)

To my wife,

Acknowledgements

After my first encounter with laser research as an undergraduate at Brandeis University with Professor I. Y. Chan, I decided science was going to be a very important part of my life. For that reason, I wanted to be trained at the best institution. I found Caltech to be the ideal research center for many more reasons than I will be able to mention here. I also found Professor Ahmed H. Zewail, who knew how to take my enthusiasm for science and amplify it by many orders of magnitude. In what seems like a very short time, I find myself writing these paragraphs, having had the greatest learning experience of my life.

Most of the learning process has been through interactions with many people at Caltech. First of all, I thank my research advisor, Professor Zewail, who provided me with financial support. He has been available for a great number of scientific discussions, and has provided me with exciting and challenging research projects. Most importantly I have learned from the trust and freedom he gave me in the design and construction of the Femtosecond Laser Facility which houses now over a million dollars in equipment. During my career at Caltech, I have had the privilege of working with very bright scientists in the Zewail research group and from all I have learned important lessons. I can only mention those that I have shared research projects with. These are: Dr. J. Spencer Baskin, Dr. Mark J. Rosker, Dr. Robert M. Bowman and Dr. Maurice H. M. Janssen. From Spencer I learned the importance of being meticulous and precise in scientific research. With Mark I shared the excitement of building the Femtosecond Laser Facility at Caltech, otherwise known in the group as "Femtoland." For two years we had the best and most intense scientific interaction. It was that dedication which made our lab a reality. I will never forget those couple of years when we were better known at Caltech as the "Marks Brothers."

From Bob I learned specific laser techniques, about polarization measurements and most of all I just enjoyed learning with him about the different research projects we shared. His friendly attitude has now spread throughout the group with noticeable results. Recently with Maurice I have shared the excitement of designing and building a molecular beam which is now an integral part of the femtosecond-laser system. I have also shared research projects with other members of the group. These are: Dr. Martin Gruebele, Dr. Amine Mokhtari, Dr. Gareth Roberts and Dr. Lawrence W. Peng. I have learned from each of them. I am especially grateful for the thorough proof-reading of this thesis by Jennifer He rek. The overall interaction with the group members, formally in group meetings and informally in the labs, corridors, and offices has been one continuous learning experience.

All of the scientific excitement would be frustration if it was not for the outstanding support of the shops and services at Caltech. Of these, Tom Dunn from the Electronic Shop deserves a medal for his dedication and interest on the success of each student at Caltech. As part of the research projects I benefited from the help of the following: Gabor Faludi and John Pirolo in the Glass Shop; Ramiro (Ray) J. Garcia, Guy L. F. Duremberg, Delmer Dill and Tony Stark from the Instrument Shop; and Jim Olson and Pavel Svitek from the Maintenance Shop. Most of the times I requested from them special attention, beyond their duty, to my projects and their dedication to them is here acknowledged.

Supporting services at Caltech have also been part of my successful career. Of these, I want to mention Fran Bennett from the Chemistry department who helped me make most of the purchases in record times. I also had interactions with the personnel in the Stock Room, Graphic Arts, Caltech Audiovisuals, the computer department the library and the managers and staff of the Athenaeum. I acknowledge their friendly service.

Since my first days at Caltech I have had one friend with whom I have taken courses, entered computer design contests, invented electronic gadgets, traded palladium futures, and have had discussions on life, science and business. His name is Terry R. Coley. Thanks to his devotion to science and his broad interests in many other fields in life, I was always able to find a person to talk about all subjects. I am sure in the future we will have more interactions. Outside Caltech I have maintained a large group of friends, none of them scientists, which have helped me grow in a non-scientific society.

Finally, I want to thank my family for their support all these years. First my parents who made me the way I am, who made me wonder how things work, and who taught me always to do a good job. They never expected I would get a Ph.D. in Chemistry, but I know they are happy and proud. Especially I want to thank my wife for having married me the way I am, without wanting to change anything. She has always supported my interest for science and has successfully allowed me to move freely in both the scientific and the non-scientific worlds. I can not thank her enough for her understanding of my love for science which at times has been obsessive. She also represents the world and culture from where I come from and without that I would have no roots.

Financial support for this research was made available through the Air Force Office of Scientific Research and the National Science Foundation. I am grateful for the Earl C. Anthony Fellowship awarded to me by Caltech.

ABSTRACT

The field of molecular dynamics emerged in the beginning of this century with the goal of understanding chemical reactions at the molecular level. The main interest was to study elementary processes such as bond rupture and bond formation as they occur on isolated molecules. Progress in that field has been directly related to technological advances. The development of the cross molecular beam techniques, for example, allowed the first observations of single molecule reactive collisions. Since the early experiments, it was recognized that the elementary steps in chemistry occurred on the time scale of a single vibration, many thought they might be unmeasurably fast. This thesis presents the first real-time observations of elementary chemical processes occurring on dissociative and bound potentials with a time resolution of 10^{-14} s.

The technique for achieving real-time monitoring of chemical reactions is Femtosecond Transition-State Spectroscopy (FTS). In essence, it involves two femtosecond (10^{-15} s) laser pulses; one to initiate the reaction and the second to probe the progress at subsequent times. In addition to the time delay between the two pulses, the experiments include different wavelengths for the pump and probe pulses. The collection of data scans as a function of time delay for each combination of pump and probe wavelengths provides sufficient information about the dynamics to allow inversion to the potential energy surface of the reaction. The potential energy surface then contains all information about the reaction.

The FTS technique has been applied with success to several systems. For repulsive states, such as the dissociation reaction of ICN, it has yielded a direct clocking of the bond rupture process and the real-time observation of transition states. For bound and quasi-bound states, the technique has allowed the real-time observation of wave packet oscillation in alkali-halides, halogens and inter-halogens. The technique has been applied to more than one dimensional systems such as HgI_2. In addition to the scalar quantities, FTS has also been successful in the measurement of vectorial quantities such as the angular momentum and torque during dissociation reactions. Of all these, and many other measurements under different conditions, only a few representative examples are discussed here. The technique is now well established and it has become an important tool for the study of molecular dynamics on bound and repulsive potential energy surfaces.

Table of Contents

Chapter I

Introduction

Understanding chemical reactions is one of the primary objectives of chemistry. Early observations showed that the time a substance required to be converted into another depended on the heat or illumination that caused the change. The relation between the amount of energy introduced versus the rate of the reaction was formulated a century ago by Arrhenius.[1] However, these macroscopic reaction rates consist of many elementary chemical transformations, such as bond rupture or bond formation, that occur on the microscopic molecules. Thus, a microscopic approach directed at isolated chemical reactions was required. The lifetime of these elementary processes was estimated to be on the order of a vibrational period, shorter than 10^{-12} seconds, and were thus considered immediate and unmeasurable. The studies included in this thesis describe the first successful measurement of the breakage of a chemical bond and the real-time observation of the intermediate elementary steps of chemical reactions and molecular dynamics with lifetimes as short as 10^{-14} seconds.

The true test for scientific understanding is to be able to predict the course of a process given the initial and final conditions. With that spirit, chemical reaction rates were measured as a function of temperature by S. Arrhenius[1] in 1889 and fitted to his now famous equation,

$$k(T) = A\, e^{-Ea/kT}, \qquad (1)$$

which related the reaction rate k to the temperature T by use of two parameters Ea and A. The "activation energy," Ea, of the reaction was determined to be the minimum energy necessary for a chemical reaction to proceed. The pre-exponential term A was associated with the frequency of reactive collisions between reactants.

Except for simple reactions governed by collisions in the gas phase or neutral species in solution, most chemical reactions did not fit the equation and revisions were made. The speculation was that some collisions were reactive while others were not. This led to the replacement of A by the product $Z{\cdot}P$, where Z is the collision number and P is the probability that the encounter is reactive. The probability or "steric" factor was found to vary from unity to 10^{-8} for increasing molecular complexity, and greater than unity for ionic reactions. All of these observations were directed to the bulk and were not concerned with the molecular processes themselves.

By 1928, chemical reactions at the molecular level were proposed to occur by a continuous change in energy as a function of internuclear distance.[2] F. London proposed that a quantum-mechanical description of a reaction could be obtained by constructing a function that mapped the potential energy of interaction between the atoms as a function of their interatomic distances. A chemical reaction must then follow a path on this "potential energy surface" (PES). The first approximate solution of this problem was achieved by H. Eyring and M. Polanyi in 1931.[3] Their method was semi-empirical since they used available spectroscopic data to define the PES of the reactants and products. They recognized the highest point in the reaction path as being virtually equivalent to the activation energy of the reaction.

Describing a chemical reaction as the flow from reactant to product on a potential energy surface rapidly led to the first successful calculation of reaction rates by H. Pelzer and E. Wigner[4] and to the development of "The Theory of Rate Processes" now known as "Transition State Theory" by H. Eyring.[5] In Eyring's formulation, a chemical reaction is described as a system "passing from one low region to another [in the PES]" and "the activated state is the highest point along the lowest pass." The activated state or activated complex, as he named it, was later renamed the "transition state" of a reaction. Results from these calculations were utilized to predict reaction rates of gas experiments where the molecular processes were averaged by equilibrium distributions of starting conditions. Chemical reactions where thus reduced to mechanical events governed by a potential as a function of the interatomic coordinates.

The first calculations aimed at understanding the microscopic reaction rates independent of the macroscopic thermal distributions were performed by J. O. Hirschfelder, H. Eyring and B. Topley in

1936.[6] Their calculations included the construction of an approximate potential energy surface for the reaction H + H_2 and the propagation of a mass point sliding on that surface, representing the translation and vibration of the atoms involved. This kind of microscopic view of chemical reactions led to the beginning of molecular reaction dynamics as a field of study. The experimental confirmation of these calculations required the development of technology capable of controlling isolated molecular collisions with a well-defined (non-equilibrium) energy. Due to technological limitations, most progress during the following two decades was theoretical in nature.

In principle, the ideal experimental setup would prepare the reactants with a given amount of energy and direct them to a region where they undergo a single collision. This kind of apparatus was developed in 1955 by S. Datz and E. Taylor; the tool was called crossed-molecular-beam reactive scattering.[7] The idea behind the molecular beam experiments of D. R. Herschbach,[8] P. R. Brooks[9] and R. B. Bernstein[10] was to prepare two beams of different reactants and cross them at a point where the reactants would have a chance to collide with each other only once. The energy and the angle of approach were the controllable variables, and the angular and energy distribution of the products were the results. From these experiments, one obtained a contour map that showed the probability of detecting the product at specific angles and velocities (energies). The contour maps provided information about the intermolecular forces of the reaction. For example, a contour could help determine what angles of approach maximize the reaction probability. Such experimental measurements were needed to confirm some of the predictions made by theoretical calculations. A theoretically calculated potential energy surface, such as the one derived by J. T. Mukerman,[11] could then be tested for reproducibility of the experimental trajectories.

Further refinement of the molecular beam technique proceeded along different routes. One concern, explored by R. B. Bernstein[10] and P. Brooks,[9] was the understanding of the role of the initial molecular orientation of the reactants prior to reaction. A second approach looked at utilizing molecular rotation as an internal clock for a reaction.[8] Since the rotation of an intermediate species takes place in 10^{-12} to10^{-13} seconds, one can judge whether the intermediate lives longer than the rotational period by looking at the contour map of the reaction. If the contour map shows forwards-backwards symmetry, it means the intermediate had time to rotate and randomize the product distribution and therefore it was long lived. Y. T. Lee's contribution to molecular beam research included the use of laser beams to prepare the initial states of the reactants precisely in certain quantum mechanical levels.[12]

The availability of tunable lasers revolutionized chemical research.[13] For the first time, a tool existed that allowed the scientist to deposit a given amount of energy into a molecule. It also allowed the characterization of the molecule after a reaction. The polarization properties could be used to study orientation, and the variable pulse duration could be utilized to study dynamics. Experimental studies of molecular reaction dynamics became very diverse, exploiting many of the characteristics of lasers and molecular beams. These advances led to the determination of the initial and final conditions of a reaction with unprecedented accuracy; electronic, vibrational and rotational quantum numbers could be measured for both reactants and products. These experiments provided product-state distributions and state-to-state reactions rates. Replacing one of the reactant beams by a laser beam in a crossed molecular experiment allowed a similar kind of angular distribution measurement of the photofragment products, as was measured by conventional molecular beam techniques. The pioneering theoretical and experimental work of R. N. Zare,[14] R. Bersohn,[15] and K. R. Wilson[16] correlated the polarization of the laser beam to the direction and speed of the products.

By the beginning of the 1980's there was a vast amount of information about the reactants and the products of several reactions, but the information about the actual process that takes place *during* could only be inferred. The reason for this lack of information was the ultra-fast lifetime of the intermediate species, the transition states of a chemical reaction. Single bond formation or rupture processes were estimated to occur on the time scale of a single vibrational oscillation (10^{-12} - 10^{-14} seconds). No technique had been devised to measure such short times. The use of electronic detectors was ruled out since their response time was three orders of magnitude slower. Furthermore, if one wanted to measure a process with a time duration of 10^{-14} seconds, one would have essentially lost the energy resolution (because of the uncertainty principle), and would then be unable to determine with certainty what was being measured.

This, seemingly hopeless, situation was best described by the feelings of some of the players in the field of molecular reaction dynamics:

S. Mukamel and J. Jortner[17] 1974
"These remarks [about the time dependence of the photodissociation process] are of academic interest only, as it is practically impossible to monitor the time evolution of the dissociation products on the time scale of the predissociation process."

R. N. Zare and R. B. Bernstein[18] 1980
"The study of chemical reaction kinetics can be likened to the task of making a motion picture of a reaction. The trouble thus far with achieving this goal seems to be the problem of too many would-be actors who strut upon the stage without proper cue and mumble their lines too rapidly to be understood--for chemical reactions occur with the ease of striking a match and at a speed so fast (on a subpicosecond time scale for the making of new bonds and breaking of old ones) as to be a severe challenge to the movie maker who would like to record individual frames."

Y. T. Lee[12] (Nobel Prize) 1986
"If the motion of individual atoms were observable during reactive collisions between molecules, it would be possible to understand exactly how a chemical reaction takes place by just following the motion of these atoms. Unfortunately, despite recent advances in microscope technology that allow us to observe the static

arrangement of atoms in a solid, we are still far from being able to follow the motion of atoms in the gas phase in real time."

Recently, there have been some successful attempts to obtain information about the transition states between reactant and product. These experiments have been reviewed by J. C. Polanyi,[19] J. L. Kinsey[20] and P. R. Brooks and R. F. Curl.[21] The principle behind these observations is the perturbation of the spectroscopy of the products of a reaction when they are still in the process of separation, within the transition state region. The broadening of the absorption and emission spectra has been known since 1915 when H. A. Lorentz interpreted the broadening of the spectral lines of the sun as arising from atoms undergoing collisions.[22] The spectral broadening during non-reactive collisions was studied in more detail by M. C. Castex[23] and J. L. Carlsten et al.[24] J. C. Polanyi observed wing emission from the dissociating of NaI[19] and J. L. Kinsey observed Raman scattering from dissociating O_3 and CH_3I and recognized the relation between detuning from resonance and the dissociation time.[20] The observation of P. R. Brooks and R. F. Curl was even more startling: using a molecular beam, they tuned a laser beam completely off-resonance from the reaction products and were able to excite a reaction intermediate and to produce a new species.[21] This observation was a direct excitation of the transition state of a reaction. More recent time-integrated observations of transition states include the bimolecular reactive scattering of J. J. Valentini[25] and the photodetachment experiments of D. M. Neumark.[26]

Unfortunately the lack of time resolution in these experiments resulted in low signal-to-noise ratios. Since the transition states have a lifetime of 10^{-13} seconds and the fluorescence lifetime of the free product is 10^{-9} seconds, the relevant signal is four orders of magnitude smaller. Only with a laser pulse shorter than the lifetime of the transition states could one capture this configuration and freeze it in a snapshot.

By 1979, A. H. Zewail coupled picosecond (10^{-12} seconds) lasers and molecular beams in order to measure dynamics which, at that time, were being inferred from frequency resolved time-integrated spectroscopy.[27] Sub-picosecond lasers (10^{-13} seconds) were then employed to measure, for the first time, a direct dissociation process.[28] The time resolution was not short enough to fully resolve the process; it was determined that the real-time observation of an elementary reaction process was within an order of magnitude improvement in time resolution. That improvement was feasible by the development of laser systems capable of generating ultrafast pulses (10^{-14} seconds), led by C. V. Shank in AT&T Bell Laboratories.[29]

In 1987, following the construction of the Caltech Femtosecond Facility, the first real-time observation of transition-states of a direct dissociation reaction was made.[30] This measurements provided the connection to the time-integrated observations of transition states.[19-21] A systematic refinement of the experimental technique and theoretical understanding led to the development of the Femtosecond Transition-State Spectroscopy (FTS) technique[31] which was utilized to clock the process of bond dissociation[32] and to observe, in real-time, the transition-states of the direct dissociation reaction of ICN.[33]

The FTS technique can be described as follows: The chemical reaction is initiated by a femtosecond pump pulse. The excitation is to a repulsive state, therefore, a bond dissociation process is initiated. As the fragments begin to separate, their dynamics are governed by the PES of the system. The transition states are spectroscopically distinguishable from reactants and products, and therefore can be probed by tuning a second femtosecond laser away from resonance of products or reactants. Wavelength tuning allows to preferential probing of the dynamics of a particular stage of the reaction by "opening a window" on the potential. The technique is two-dimensional, time and wavelength, and each measurement is a "snapshot" taken with a given probe wavelength at a series of pump-probe time delays. Further refinement of the ICN theoretical interpretation has been performed, considering the classical, semiclassical and quantum mechanical descriptions of the reaction (see chapter II).[34]

FTS experiments on NaI by T. S. Rose and M. J. Rosker[35] led to the observation of resonances during bond dissociation due to curve crossing. These experiments allowed the calculation of the Landau-Zener crossing probabilities of the initially prepared wave packet. While the crossing of the ionic and the covalent surfaces was known, the real time observation was striking. Again for this reaction, monitoring the free product led to the observation of a build-up with steps corresponding to the escape of the covalent atoms from the Landau-Zener crossing. When the probe wavelength was tuned to the transition states, the actual oscillations inside the well created by the curve crossing could be observed in real time. Classical, semiclassical and quantum mechanical calculations have come very close to describing the observed experimental data.[36] Current efforts in our group by P. J. Cong and A. Mokhtari have concentrated on further refinement of the experimental data and the theoretical interpretation of the long time behavior of the transients, the anisotropy of the reaction and the effect of high intensity of the laser field on the reaction.[37]

After the initial success of the FTS technique on diatomics and quasi-diatomics, HgI_2, a triatomic, was studied.[38,39] Upon photolysis, three reactions can be expected,

$$HgI_2 \rightarrow [HgI_2]^{\ddagger *} \rightarrow HgI + I,$$
$$HgI_2 \rightarrow [HgI_2]^{\ddagger *} \rightarrow HgI + I^*, \text{ and}$$
$$HgI_2 \rightarrow [HgI_2]^{\ddagger *} \rightarrow I + Hg + I.$$

The three possibilities were observed, including the observation of the nascent HgI fragment rotational and vibrational motion. Tuning the probe wavelength into the transition state region allowed the observation at the very first stages of the dissociation. Even at those early times, the distinction of the I and I* channels could already be made. The ability to discriminate between the two channels was achieved by realizing that the I channel led to an HgI fragment with higher vibrational energy. By discrimination of the fluorescence arising from highly excited HgI fragments, one could observe the I* channel and vice versa. The theoretical interpretation of these experiments was aided by the classical theory and by semiclassical trajectories and quantum mechanical calculations by M. Gruebele et al., including predictions of the fluorescence spectrum of the nascent HgI fragment.[39] Further refinement of the theory, including quantum calculations of the wave packet dynamics on the two dimensional surface, are being carried out in our group by M. Gruebele and G. Roberts.[40]

The possibility of utilizing the polarization properties of the femtosecond lasers used in the FTS techniques was explored in a study on the role of alignment in the ICN and HgI_2 reactions.[41] While most of the interpretation of the previous FTS experiments had focused on the scalar properties of the dissociation (increasing bond separation), the focus of this study was on the vector properties, that is, the alignment and angular momentum of the nascent fragments during dissociation. The interpretation of the results was based on a simple theoretical description[42] based on the theory developed by P. M. Felker and J. S. Baskin[43] for pure rotational coherence in photoexcitation experiments. The results showed the initial rotation of the CN or HgI fragments from the very first angstroms of the reaction. For ICN, the dissociation time is comparable to the alignment (~200 fs) and therefore alignment differences between parallel and perpendicular probing could only be observed for low rotational energy levels.

The power and versatility of the FTS technique had been demonstrated in the previous experiments; it was then directed towards the B bound state of molecular iodine.[44] Since the femtosecond probing laser is sensitive only to a restricted region of the bound well dictated by the Franck-Condon overlap to the upper excited state, the signal shows an oscillatory behavior which corresponds to the transient behavior of the wave packet in the well. As expected, an increase in the pumping energy leads to excitation of an upper section of the anharmonic well and thus longer oscillation times are observed. The alignment analysis on these FTS experiments yielded pure rotational coherence recurrences due to the different rotational constants for each of the vibrational levels accessed.[45] The information contained in this temporal data has been Fourier Transformed and the values correspond to the vibrational and rotational levels found from high resolution spectroscopy.[46]

These numbers were then used with a quantum mechanical inversion method by M. Gruebele et al.[46] to obtain a potential energy surface that was very close to the one calculated by fitting hundreds of lines from high resolution spectroscopy. The experiments included data for very high excitation energies that led to dissociation of the two iodine atoms.

Current efforts in our laboratories include the study of bound and reactive potential energy surfaces. A molecular beam machine has recently been built in the laboratory and will add the capability of time-of-flight mass spectroscopy to the arsenal of techniques used in conjunction with the femtosecond laser system. There is interest in the group to extend these techniques to the condensed phase and to the study of bimolecular reactions.[47] These goals are currently being pursued with the characteristic excitement of this research group; results can be expected soon.

The body of the thesis is organized into four chapters. In Chapter II, theoretical background of the FTS technique is derived, first from a simple classical mechanics approximation which is sufficient to describe many of the experimental results, then further refinement is added and the theory is compared to full quantum mechanical calculations. Chapter III describes Caltech's Femtosecond Laser Facility with particular emphasis on the generation of ultrafast pulses, their amplification and characterization. Chapter IV contains the applications of the FTS technique to the (i) observation of transition-states in the dissociation reaction of ICN, (ii) dissociation processes of more than one coordinate, the dissociation of HgI_2, (iii) FTS of bound states applications to molecular iodine, and (iv) the role of alignment in FTS experiments.

References

1. S. Arrhenius, Z. Physic. Chem., 4, 226, (1889).
2. F. London, Probleme der modernen Physic, Somerfeld Festschrift, 104, (1928); F. London, Z. Elektrochem., 35, 552, (1929).
3. H. Eyring and M. Polanyi, Z. Physic. Chem., B, 12, 279, (1931).
4. H. Pelzer and E. Wigner, Z. Physic. Chem., B, 15, 445, (1932).
5. H. Eyring, J. Chem. Phys., 3, 107, (1935); see also S. Glasstone, K. J. Laidler, and H. Eyring, The Theory of Rate Processes, McGraw-Hill, New York, (1941).
6. J. O. Hirschfelder, H. Eyring and B. Topley, J. Chem. Phys., 4, 170, (1936).
7. S. Datz and E. Taylor
8. D. R. Herschbach, Adv. Chem. Phys., 10, 319, (1966); D. R. Herschbach, Faraday Discuss. Chem. Soc., 55 233, (1973); D. R. Herschbach, Angew. Chem. Int. Ed. (Eng.), 26, 1221, (1987).
9. P. R. Brooks, Science, 193, 11, (1976).
10. R. B. Bernstein, in Atom-Molecule Collision Theory. A Guide for the Experimentalist, Plenum, New York, (1979); R. B. Bernstein, Chemical Dynamics via Molecular Beam and Laser Techniques, Oxford Univ. Press. New York, (1986).
11. J. T. Mukerman, in Theoretical Chemistry-Advances and Perspectives, H. Eyring and D. Henderson, Eds., 6A, pp 1-77, Academic Press, New York, (1981).
12. Y. T. Lee in Atomic and Molecular Beam Method, G. Scoles and U Buck, Eds. Oxford Univ. Press, New York, (1982); Y. T. Lee, Angew. Chem. Int. Ed. (Eng.), 26, 936, (1987).
13. See the special issue on Laser Chemistry: A. H. Zewail, V. S. Letokhov, R. N. Zare, R. B. Bernstein, Y. T. Lee and Y. Ron Shen, Physics Today, 33(11), 25, (1980).
14. R. N. Zare, Mol. Photochem., 4, 1, (1972); R. N. Zare and P. J. Dagdigian, Science, 185, 739, (1974); R. N. Zare, Angular Momentum, Willey, New York, (1988).
15. M. J. Dzvonik, S. Yang and R. Bersohn, J. Chem. Phys., 61, 4408, (1974).

16. S. J. Riley and K. R. Wilson, Faraday Disc. Chem. Soc., 53, 132, (1972); G. E. Busch and K. R. Wilson, J. Chem. Phys., 56, 3626, (1972).
17. S. Mukamel and J. Jortner, J. Chem. Phys., 61, 5348, (1974).
18. R. N. Zare and R. B. Bernstein, Physics Today, 33(11), 43, (1980).
19. H. -J. Foth, J. C. Polanyi and H. H. Telle, J. Phys. Chem., 86, 5027, (1982); Angew. Chem. Int. Ed. (Eng.), 26, 952, (1987).
20. D. Imre, J. L. Kinsey, A. Sinha and J. Krenos; J. Phys. Chem. 88, 3956, (1984).
21. P. Hering, P. R. Brooks, R. F. Curl, R. S. Judson and R. S. Lowe, Phys. Rev. Lett., 44, 687, (1980).
22. H. A. Lorentz, Proc. Acad. Sci. Amsterdam, 18, 154, (1915).
23. M. C. Castex, J. Chem. Phys., 66, 3854, (1977).
24. J. L. Carlsten, A. Szöke and M. G. Raymer, Phys. Rev. A, 15, 1029, (1977).
25. J. -C. Nieh and J. J. Valentini, Phys. Rev. Lett., 60, 519, (1988).
26. R. B. Metz, T. Kitsopolous, A. Weaver and D. M. Neumark, J. Chem. Phys., 88, 1463, (1988).
27. A. H. Zewail, Laser Chem., 2, 55, (1983).
28. N. F. Scherer, J. L. Knee, D. D. Smith and A. H. Zewail, J. Phys. Chem., 89, 5141, (1985).
29. C. V. Shank, Science, 233, 1276, (1986) and references there in.
30. M. Dantus, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 87, 2395 (1987).
31. M. J. Rosker, M. Dantus and A. H. Zewail, J. Chem. Phys. 89, 6113 (1988).
32. M. J. Rosker, M. Dantus and A. H. Zewail, Science, 241, 1200 (1988).
33. M. Dantus, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 89, 6128 (1988).
34. S. O. Williams and D. G. Imre, J. Phys. Chem., 92, 6648, (1988); I. Benjamin and K. R. Wilson, J. Chem. Phys., 90, 4176, (1989).
35. T. S. Rose and M. J. Rosker and A. H. Zewail, J. Chem. Phys. 88, 6672, (1988); T. S. Rose and M. J. Rosker and A. H. Zewail, Chem. Phys. Lett., 146, 175, (1988).

36. V. Engel, H. Metiu, R. Almeida, R. A. Marcus, A. H. Zewail, Chem. Phys. Lett., 152, 1 (1988).
37. P. J. Cong, A. Mokhtari and A. H. Zewail, Chem. Phys. Lett., 172, 109, (1990).
38. R. M. Bowman, M. Dantus and A. H. Zewail, Chem. Phys. Lett., 156, 131, (1989).
39. M. Dantus, R. M. Bowman, M. Gruebele and A. H. Zewail, J. Chem. Phys., 91, 7437, (1989).
40. M. Gruebele and G. Roberts and A. H. Zewail, Philos. Trans. R. Soc., (1990).
41. M. Dantus, R. M. Bowman, J. S. Baskin and A. H. Zewail, Chem. Phys. Lett., 159, 406, (1989).
42. A. H. Zewail, J. Chem. Soc. Faraday Trans., 2, 85, (1989).
43. P. M. Felker, J. S. Baskin and A. H. Zewail, J. Phys. Chem., 90, 724, (1986); P. M. Felker and A. H. Zewail, J. Chem. Phys., 86, 2460, (1987); J. S. Baskin, P. M. Felker and A. H. Zewail, J. Chem. Phys., 86, 2483, (1987).
44. R. M. Bowman, M. Dantus and A. H. Zewail, Chem. Phys. Lett., 161, 297, (1989).
45. M. Dantus, R. M. Bowman and A. H. Zewail, Nature, 343, 737, (1990).
46. M. Gruebele, G. Roberts, M. Dantus, R. M. Bowman and A. H. Zewail, Chem. Phys. Lett., 166, 459, (1990).
47. N. F. Scherer, L. R. Khundkar, R. B. Bernstein and A. H. Zewail, J. Chem. Phys., 87, 1451, (1987).

Chapter II

Femtosecond Transition-State Spectroscopy: A Theoretical Description

The first real-time measurement of a direct dissociation reaction spurred the development of a theory capable of reproducing the experimental data.[1] Such a theory would have to be realistic enough to provide insight into the physics of the problem, but simple enough to be conceptually transparent. The chosen approach by Bersohn and Zewail was a classical approximation of a reduced mass sliding on a simple exponential-repulsion potential.[2] The model, however simple, was able to reproduce the shape of the transients. Following the ideas of this classical picture, Bernstein and Zewail proposed a method by which the data from an FTS experiment could be directly inverted to the potential energy surface of the reaction.[3] This method has been used to invert the ICN experimental data and its success is commensurate with the quality of the classical approximation.[4] Mainly it provides a very simple procedure that can be used to obtain an approximate potential energy curve for a reaction without having to do a complete quantum mechanical simulation.

In fact, the success of the simple classical approach has been so remarkable, that its derivation and implications to experimental results are included in this thesis. Results from the simple classical model are compared to the experimental results, and are later compared to full quantum mechanical models[5-7] which take into account the three main processes: preparation of the initial wave packet by the pump laser, propagation of the wave packet along the repulsive potential, and probing the wave packet after a specified delay time. A comparative study has been carried out by Fried and Mukamel.[6] In our group, a comparison of the full quantum calculations and the classical model is also in progress, taking the available experimental FTS data as the basis for the contrast; preliminary results show excellent agreement.[8]

Beyond the simple one dimensional calculations, we have extended the classical model to include the possibility of rotation of the fragments. Direct comparison with experimental transients has allowed estimates of the initial changes in the angular momentum even before the fragments are free. The classical model has also been applied to the quasi-bound potentials such as the NaI[9] and to bound states such as the B state of molecular iodine.[10]

The organization of this chapter is as follows: First, the concept of the FTS technique is presented, including the effect of the pump and probe experiments on a simple repulsive potentials. Second, the classical propagation of a particle with the appropriate reduced mass is considered and included in the FTS formulas in order to introduce the time dependence. This section includes comparison to experimental data and explores the dependence of the signal to the characteristics of the potential energy surfaces considered. Third, the inversion procedure is considered from the classical model point of view and its applications to experimental results are considered. Fourth, the classical model is compared to full quantum calculations. Fifth, the length of the pulses and its effect on FTS experiments is explored. Finally, we conclude with present extensions of this model to other experiments where its simplicity has yielded qualitative, and quantitative, information about the molecular dynamics under investigation.

1. The concept of FTS

The FTS technique is a two-dimensional experimental technique in which, at time zero, a femtosecond pump laser excites a molecule AB, originally in its ground electronic state, to a dissociative excited state. Due to the repulsion in the dissociative state, the fragments A and B begin to separate until both of the fragments can be considered free from one another. At this point, the dissociation process is considered complete (more about the dissociation time will be considered in Section 2.1). The analytical tool that monitors the progress of the reaction is a femtosecond probe laser which is tuned to a resonance wavelength of the B fragment. The probe excitation on the B fragment promotes it to a fluorescent state. In theory, monitoring the absorption of B or the fluorescence of the excited B fragment (B*) would be equivalent. Experimentally, detection of the laser induced fluorescence of B* is the most sensitive choice since the experiment is virtually background free.

In Figure 1, the relevant potential curves are shown schematically in order to introduce the FTS terminology. The pump laser with wavelength λ_1 excites the molecule AB to the repulsive potential $V_1(R)$ at time zero. The internuclear separation between A and B is taken to be their ground state equilibrium bond distance R_e. After a relatively long time delay, the probe laser, with a wavelength λ_2^∞ which matches a resonance of the B fragment (on-resonance probing), is absorbed and generates B* which is then detected. The fluorescent B* state correlates to the potential energy surface (PES) $V_2(R)$ of a highly excited AB** molecule. Probing is not restricted to long times only; intermediate times can also be probed. Tuning the probe to a longer wavelength λ_2^* induces a transition from the transition states of the reaction, $[AB]^{\ddagger *}$, possible before the reaction is complete. Note that a maximum signal should be expected when the fragments are traversing the region R* for which $\Delta V(R^*) = V_2(R^*) - V_1(R^*)$ equals $1/\lambda_2^*$.

Consider, at first, laser pulses that have a very narrow frequency bandwidth. The absorption as a function of internuclear distance, A(R), for a hypothetical $[AB]^{\ddagger *}$ transition state is then simply

$$A(R,\lambda_1,\lambda_2^*) = \begin{bmatrix} 1\,; & \Delta V(R) = 1/\lambda_2^* \\ & \\ 0\,; & \Delta V(R) \neq 1/\lambda_2^* \end{bmatrix} \qquad (1)$$

For the case schematically considered in Figure 1, where the potential curves are approximated to exponential repulsion curves of the form

$$V_i(R) = V_i(R_e)\exp\{-(R-R_e)/L_i\} + C_i\,, \qquad (2)$$

for i = 1 and 2, and the length parameters L_1 and L_2 are different, there will be an absorption only for values of R that satisfy the condition of Equation (1). The points will be R*, as expected. The parameter C is equivalent to the energy displacement of the B and B* stationary states, therefore, $C_2 - C_1 = 1/\lambda_2^{\infty}$. On-resonance probing at λ_2^{∞}, will result on a step function, for infinitely narrow pulses, which turns on at R equals infinity, when both potentials have reached their asymptotic value.

A not so trivial case that deserves a pictorial representation is the case of a probe laser with a square frequency bandwidth with FWHM γ. The absorption condition is basically an extension of the previous case, where

$$A(R,\lambda_1,\lambda_2^*) = \begin{cases} 1 \ ; \left|\Delta V(R) - 1/\lambda_2^*\right| \leq \gamma/2 \\ 0 \ ; \left|\Delta V(R) - 1/\lambda_2^*\right| > \gamma/2 \end{cases} \qquad (3)$$

but the condition now includes all points of the potential within the "window" opened on the potential by the probe pulse. The absorption as a function of R is depicted in Figure 2 for on- and off-resonance probing.

In order to illustrate how the probe laser "opens a window" on the potential, a three dimensional plot is considered in Figure 3. The three dimensions are the internuclear distance in Å, the potential energy or spectral coordinate in wavenumbers, and a relative intensity coordinate which relates the probe intensity to the absorption. Note that the probe pulses have a spectral FWHM of γ and a central wavelength λ_2^* which is off-resonant for every R position by an amount:

$$x = \Delta V(R) - 1/\lambda_2^*. \qquad (4)$$

With this kind of plot, the relation between probe, PES, and A(R) is simplified. In the following section, a similar plot will be used to show the time dependence of the signal.

The spectral shape of the probe pulse essentially determines the absorption as a function of R. For a Gaussian pulse, which is experimentally realistic, A(R) takes on a Gaussian shape which includes the spectral matching condition as before. Keeping the

FWHM of the pulse equal to γ by including the factor (4ln2), the new equation takes the form

$$A(R,\lambda_1,\lambda_2^*) = \exp\left(\frac{-(4\ln 2)\, x^2}{\gamma^2}\right) \qquad (5)$$

and is plotted in Figure 4 for on- and off-resonance probing. A similar expression can be derived for a Lorentzian probe with FWHM γ,

$$A(R,\lambda_1,\lambda_2^*) = \frac{1}{1+\left(4x^2/\gamma^2\right)} \qquad (6)$$

its corresponding effect is also shown in Figure 4. In this case a factor of 4 was added to keep the FWHM equal to γ. Notice the Lorentzian function has wings that extend further than those of the Gaussian.

So far, we have introduced all of the parameters needed to describe an FTS experiment, except for the time dependence which is discussed in the next section. The expressions derived for square, Gaussian and Lorentzian probe pulses are generally applicable to any pair of potentials V_1 and V_2 and are independent of the time characteristics of the pump and probe pulses as well as the reduced mass of the molecule in question.

2. Introducing the time dependence

Since the potential curve at the initiation of the reaction is at its steepest point, the projection of the pump laser bandwidth to the potential is a very narrow function of R. Thus, taking an average R value is a reasonably good approximation in the spirit of a simple classical mechanical model. The initial localization in R makes the classical FTS model similar to a single trajectory calculation. However, the time characteristics of the pump and probe pulses are not be neglected, and therefore the initial R localization is not a localization in time.

In principle, the dissociation reaction of AB is a function of the initial excitation energy and the interatomic PES. The molecular dynamics of this reaction, M(t), can be described classically as if a particle with the appropriate reduced mass slides along the path of steepest descent. The temporal behavior of the pump process, $I_{pu}(t)$, has the effect of convolving the pure molecular dynamics, M(t). This convolution smears the initial time localization of the particle. The probe pulse, $I_{pr}(t)$, has a similar effect of convolution and the observed signal is then

$$S(t) = I_{pu}(t) * M(t) * I_{pr}(t). \qquad (7)$$

Making use of the commutative and associative properties of these functions we can simplify S(t) in the following manner:

$$S(t) = M(t) * C(t) , \qquad (8)$$

where

$$C(t) = I_{pu}(t) * I_{pr}(t) \qquad (9)$$

is the cross-correlation function between the pump and probe pulses. The implication of Equation 8 is that the temporal convolution of the pump and probe processes can be ignored while one calculates the molecular dynamics, and can be included once M(t) has been calculated.

The derivation of a formula that can describe the time-dependent absorption as a function of pump and probe wavelengths similar to Equations 5 and 6 is very simple. The only requirement is the substitution of $\Delta V(R)$ by $\Delta V(t)$ when calculating x (from Equation 4): everything else remains the same. In general,

$$A(t) = F(x), \qquad (10)$$

where F is the spectral shape of the probe laser and x is defined by Equation 4 using $\Delta V(t)$.

The calculation of $\Delta V(t)$, given $\Delta V(R)$, requires an expression for R as a fraction of time, R(t). Such an expression can be calculated making use of the classical equations of motion. We start by considering that at time zero, the total energy of the reaction is E. As time evolves, the potential energy available is converted into kinetic energy. In mathematical terms,

$$E = V_1(R) + (1/2)\,\mu(dR/dt)^2, \qquad (11)$$

where the R coordinate is simply taken to be zero at the starting point of the trajectory ($R = R' - R_e$), so that $V(0) = E$ and μ is the molecular reduced mass.

Separation of variables leads to the following expression which can be integrated analytically or numerically to obtain R(t);

$$dt = \{2[E-V_1(R)]/\mu\}^{-1/2}\, dR. \qquad (12)$$

Since, for the example at hand, V(R) is assumed to be a simple exponential-repulsion, as determined from Equation 2, the integration can be done analytically. Before integration, however, we can simplify the expression further. The terminal velocity reached by the fragments after dissociation, v_t, can be related to the total energy of the reaction by the expression,

$$v_t = (2E/\mu)^{1/2}\,, \qquad (13)$$

and so the expression becomes

$$\begin{aligned} dt &= \{2E[1-\exp(-R/L_1)]/\mu\}^{-1/2}\, dR \\ &= \{v_t[1-\exp(-R/L_1)]^{1/2}\}^{-1}\ dR, \qquad (14) \end{aligned}$$

which now includes the shape of the potential. The resulting expression after integrating is

$$R(t) = -L_1 \ln\{\ \mathrm{sech}^2(v_t t/2L_1)\}, \qquad (15)$$

or

$$\exp(-R/L_1) = \mathrm{sech}^2(v_t t/2L_1), \qquad (16)$$

and with this expression, we can easily convert between $\Delta V(R)$ and $\Delta V(t)$.

To illustrate the dynamics on the dissociative potential, we plot a three-dimensional graph relating the potential energy V, the internuclear distance R and the time t; in short, VRT. Figure 5 shows the VRT curve calculated in relation to the ICN dissociation dynamics.[4] This graph is useful in illustrating how fast the fragments reach their terminal velocity.

Given equations for $A(t,\lambda_1,\lambda_2^*)$, we are now ready to analyze the behavior of the model with respect to the experimentally variable parameters, such as time delay, wavelength of the lasers, and the probe bandwidth. As mentioned before, we are allowed to neglect the temporal width of the lasers, provided we convolve A(t) with the appropriate cross correlation afterwards. For this reason, we explore the results of the model independent of convolution and later determine its effects.

2.1 On-resonance clocking experiments

As shown in Figure 4, A(R) for on resonance probing reaches a maximum after a given R separation. Determining how long it takes to reach this condition is analogous to measuring the dissociation time of the reaction. For the most general treatment, consider the absorption of the probe A(t) to be dependent on the shape of the probe laser as mentioned for A(R). When the pulse has a spectral shape $F(\nu)$, and F is any function symmetric about the origin with a maximum at F(0) and a FWHM of γ, we recover Equation 10. F can be a Gaussian, Lorentzian or any other shape which satisfies the three stated conditions, and x can be distinguished as

$$x = \Delta V - 1/\lambda_2^* \tag{17}$$

for ΔV a function of t or R. The condition that F has a FWHM of γ implies that $F(\gamma/2) = 1/2$. This new result also implies that the absorption equals 1/2 when the detuning equals half of the spectral FWHM of the probe pulses. Note that this width, which arises purely from the spectral shape of the probe laser, is not related to the temporal convolution of the pump and probe lasers which has not been included in this model yet.

In a sense, the FTS measurements can be considered a two step process. Within this classical model, the first step places a particle, with a sharp R and energy distribution, on the dissociative potential. The second step utilizes an analytical tool, the probe laser, which has a tunable range and selectivity for different species. It is to be expected that all FTS measurements depend on the characteristics of this analytical tool. Equation 10 makes this explicit dependence. This fact must not be considered a shortcoming of the FTS technique because F, the spectrum of the probe pulse, can be experimentally obtained, and the information about the PES can be extracted by an inversion procedure which will be discussed in Section 4.

From the previous observations, it will be clear that one can determine with accuracy the point on the PES when the particle reaches the condition

$$|\Delta V - 1/\lambda_2^*| = -\gamma/2. \qquad (18)$$

This is particularly important for on-resonance probing at λ_2^∞ because it provides a phenomenological measure of the bond dissociation time which depends explicitly on the analytical tool which measures it. The previous equation can be used to find this time, based on the time dependence of $\Delta V(t)$. Thus in general, a clocking experiment will yield a "dissociation time" $\tau_{1/2}$, which satisfies the condition

$$\Delta V(\tau_{1/2}) = 1/\lambda_2^\infty - \gamma/2. \qquad (19)$$

When using a model with two different potentials V_1 and V_2, $\tau_{1/2}$ can be obtained by numerical calculation. However, a reasonable approximation can be made about $V_2(R)$ at long R that can substantially simplify Equation 19. If, at long R's, $V_2(R)$ reaches its asymptotic value, C_2, faster than $V_1(R)$, then,

$$\Delta V(\tau_{1/2}) = C_2 - V_1(\tau_{1/2}). \qquad (20)$$

Since $C_2 - C_1$ equals$1/\lambda_2^\infty$, then $V_1(\tau_{1/2})$ equals $\gamma/2$. Using Equation 16 and $V_1(t)$ from Equation 2, we obtain the condition

$$E\ \mathrm{sech}^2(v_t\tau_{1/2}/2L_1) = \gamma/2. \qquad (21)$$

Solving for $\tau_{1/2}$, the expression becomes

$$\tau_{1/2} = (L_1/v_t)\ln(8E/\gamma), \qquad (22)$$

which is an explicit function for the dissociation time and depends on the natural logarithm of the inverse of the FWHM of the spectral bandwidth of the probe laser, the analytical tool. Notice that $\tau_{1/2}$

does not depend on the functional form of the pulse shape, $F(\nu)$, only its FWHM. Figure 6 shows a plot of the γ dependence of $\tau_{1/2}$ measurements, for $L_1 = 0.8$ Å, $v_t = 0.026$ Å/fs and $E = 6000$ cm^{-1}. The corresponding temporal width of the pulses is calculated for Gaussian transform limited pulse satisfying the condition,

$$\Delta\nu\Delta t = 0.4413, \qquad (23)$$

for the frequency ($\Delta\nu$) and temporal (Δt) FWHM's of the laser. Figure 6 also shows the dependence of $\tau_{1/2}$ on the total energy of the reaction. It is interesting to consider how increasing the initial total energy changes $\tau_{1/2}$; the dependence on the total energy E enters explicitly and should also be included in the calculation of v_t from Equation 13. Figure 7 shows A(t)'s for Gaussian probe widths of 366, 118 and 74 cm^{-1} which correspond to transform limited pulses of 40, 125 and 200 fs, respectively. For these calculations; $V_2(R)$ was assumed flat, $V_1(0) = 6000$ cm^{-1} and $L_1 = 0.8$ Å. When $V_2(R)$ is not flat, $\Delta V(t)$ depends on both potentials and must be taken into account. For illustration, a set of potential parameters with a not flat $V_2(R)$ is also used in this simulations. The values (summarized in Table 1) were obtained by the integration of the absorption spectrum of ICN and are used in most theoretical calculations.[5-7] Figure 7 shows transients for both potential parameters, flat and not flat. Note that, for the not flat potentials, the length parameter $L_1 = 0.2$ Å is smaller than L_1 for the flat potentials, thus the dynamics are much faster.

2.2 Off-resonance experiments

Making the analytical probe sensitive to the transition states of the reaction is equivalent to tuning the probe laser to longer wavelengths, λ_2*, as shown schematically in Figure 1. Since λ_2* is most sensitive to the transition state region located at R*, A(R) is expected to have a maximum at R*. In terms of time, the transient signal reaches a maximum after a time delay t*, for $\Delta V(R^*) = \Delta V(t^*)$. A series of measurements for different λ_2*, say

$A(t,\lambda_1,\lambda_{2(a)}^*,\lambda_{2(b)}^*,\ldots,\lambda_2^\infty)$ can essentially map $\Delta V(t)$ in the transition state region. Figure 8 shows calculated transients for probing wavelengths 388.9, 389.8, 390.4, and 391,4 nm, using the parameters for V_2(flat) in Table 1 and γ set to 118 cm^{-1}. Notice that the further the probe laser is tuned from resonance, the sharper A(t) is. This effect is due to the increasing slope of the potential at short R's. For the calculations shown in Figure 8, V_2 was assumed flat for simplicity. Similar calculations are shown in Figure 9 using the potential parameters for V_2(not flat), in Table 1, for a probe wavelength of 389.5 nm. Note throughout this section that the length parameter is different for the two sets of potentials.

Besides the maximum reached at t*, off-resonance experiments are characterized by an asymptotic level, $A(\infty)$. The further off-resonance, the smaller $A(\infty)$. For the general Equation 10,

$$\begin{aligned} A(\infty) &= F\{\Delta V(\infty) - 1/\lambda_2^*\} \\ &= F\{1/\lambda_2^\infty - 1/\lambda_2^*\} \\ &= F(\Delta), \end{aligned} \tag{24}$$

where Δ is the detuning from resonance. Experimentally, Δ is measured accurately for each experiment independent from the transient. Thus, $A(\infty)$ provides a separate check on the theoretical model. So far we have neglected the temporal widths of the pump and probe pulses and their effect on the signal. It is to be expected that the further off-resonance the probe laser is tuned, the more severe the convolution effect is. This behavior is due to the fact that A(t) is sharper for off-resonance experiments, can become much narrower than the cross correlation between pump and probe pulses.

2.3 *The effect of convolution*

The combined temporal smearing effect of the pump and probe pulses, that is the cross correlation, is routinely determined for each FTS experiment. So far in this simple model, we have shown that it is equivalent to either calculate the molecular dynamics and then convolve with the cross correlation or to convolve independently with the pump and probe pulses. Thus, the expected signal is

$$S(t) = A(t) * C(t), \tag{25}$$

where A(t) contains the spectral characteristics of the probe pulse and the molecular dynamics M(t), and C(t) is the cross correlation of the pulses. Notice that the absolute amplitude of S(t) is not determined since it depends on laser intensities, absorption cross sections, pulse intensities, concentrations, path lengths, detection efficiencies and a host of other experimental parameters that can be effectively lumped into a constant. This constant is taken to be unity.

The top of Figure 10 shows the effect of convolution for clocking experiments. Since A(t) for clocking experiments is symmetric about $\tau_{1/2}$, convolution with broader cross correlations does not affect the measurement. However, it is important to qualify this statement. The clocking experiment depends on the spectral width of the probe pulse γ; as long as γ is kept constant, $\tau_{1/2}$ is unchanged. This is an important fact that could lead to misunderstandings. When transform limited pulses are used, increasing the time width of a pulse in fact reduces the spectral width proportionally (see Equation 23 for Gaussian transform limited pulses). It is this change in spectral width which will change the $\tau_{1/2}$ measurements on the bottom of Figure 10.

For off-resonance calculations, the effect of convolution is not as straightforward as for the clocking experiments. For the cases where there is a very small asymptotic level of less than 1%, the convolution mainly adds a substantial width to the transient without displacing the center of the peak at t*. When the asymptotic level is greater, the t* value is displaced towards longer times upon convolution due to an asymptotic level increase. Figure 11 shows these observations. We will consider how to extract information from an FTS experiment given that the values of t* are shifted by the convolution of the pump and probe pulses.

Mathematically, if the resulting signal is the result of convoluting the molecular dynamics by the cross correlation of the pump and probe pulses, then the signal must be deconvoluted by that same function in order to obtain the desired dynamics. The process of deconvolution is simple, but not very stable, and can be problematic for noisy data sets. Equivalently, one can generate a possible A(t) and convolve it with the experimentally determined cross correlation; the result can then be compared to the experiment. For off-resonance experiments, the asymptotic level can be determined by the detuning of the probe laser. When determining the effect of convolution, one must be able to find a deconvolved transient that has the appropriate asymptote for the known experimental detuning. This procedure has been utilized in order to find the best fitting parameters for the experimentally obtained data in Figure 12. Since the experimental cross correlation was used to convolve the simulations, the shape and noise of the cross correlation was transferred to the transients. As can be seen in Figure 12, the simple classical model can be used to fit the experimental data.

It is important to consider the question of uniqueness of the fit to the experimental data. As mentioned earlier, the clocking experiments are independent of both the pulse shape and the time widths of the pulses; they depend only on the spectral width of the probe pulse and are therefore very reliable measurements. Given a model for the potentials involved, the clocking experiments can be used to determine the parameters that fit the observed dynamics. A series of measurements of $A(\tau_{1/2},\lambda_{1(a)},\lambda_{1(b)},\ldots,\lambda_2^{\infty})$ as a function of increasing pump wavelength with constant probe wavelength (on-resonance) provides additional clocking experiments that can ascertain the contributions of V_1 and V_2 to the observed dynamics. The total set of FTS experiments, $A(\tau_{1/2}, \lambda_{1(a)}, \lambda_{1(b)}, \ldots, \lambda_{2(a)}^*, \lambda_{2(b)}^*, \ldots, \lambda_2^{\infty})$ allows mapping of the potential energy surfaces involved. For a better mapping technique, the direct inversion procedure can be used. The main advantage of the direct inversion is that no functional form has to be assumed about the surfaces *a priori.*

3. The inversion of FTS data to the PES

The following discussion is based on References 2 and 3 which are the initial presentation of this classical model. This procedure is of importance because real-time measurements, such as the FTS experiments described in this thesis, may be the best technique to extract information about potential energy surfaces, especially for dissociation reactions where the intermediates are not sufficiently long lived and other spectroscopic techniques cannot be used. As mentioned earlier, an FTS experiment provides a measure of the absorption (or laser induced fluorescence) of the transition-state species as a function of time. The measurements depend on two main variables: the pulse characteristics, such as time and frequency width which are experimentally determined and can be considered as known; and the potential energy change as a function of time, which determines the spectroscopy of the intermediate species and is the unknown.

In the following discussion we analyze the treatment of a series of FTS experiments. The available data are a series of measurements $A(t, \lambda_{1(a)}, \lambda_{1(b)}, ..., \lambda_{2(a)}^*, \lambda_{2(b)}^*, ..., \lambda_2^\infty)$ which have been deconvolved by the cross correlation of the pump and probe pulses C(t). The probe pulse is also spectroscopically well characterized and its shape is F(ν). In principle, the inversion procedure is nothing more than taking Equation 10 and solving for x:

$$x = F^{-1}(A(t)), \tag{26}$$

for each data set. Here F^{-1} is the inverse function of F, which is the spectroscopic shape of the probe pulse, and x is the detuning of the probe as defined by Equation 4. When F is a well known function such as a Gaussian or a Lorentzian, its inverse function can be found analytically. When F(ν) cannot be successfully approximated by a simple function, the experimentally determined F(ν) can be used to numerically invert the data.

The concept of inversion is best described by Figure 13, which is similar to Figure 3, except that the internuclear spacing coordinate, R, is now replaced by the delay time. The FTS transient is a mapping of the pulse spectrum on the potential energy surface. In this case, the potential energy surface corresponds to $\Delta V(t)$. Note once again that the asymptotic level of a transient is directly related to the detuning of the probe pulse. By application of Equation 26 or numerical methods we obtain x(t), which is related simply to $\Delta V(t)$. Calculating $\Delta V(t)$ for a series of transients obtained at different probe wavelengths helps refine the resulting curve, since these transients are sensitive to different regions of the PES. For the simple case where the upper potential V_2 is considered flat, the inversion procedure yields $V_1(t)$ directly. In Figure 14, three transients are inverted and the resulting potentials are shown.

When the upper potential is not flat, we can at least determine $\Delta V(t)$. In order to obtain $V_1(t)$ we need to analyze the transients obtained for different pump energies. Since the dynamics depend on R(t), which is determined by V_1, the complete set of FTS experiments contains enough information to calculate $V_1(t)$, and by subtraction, $V_2(t)$. In practice, it is useful to approximate the shapes of the potentials to a simple functional form such as exponential-repulsion in order to obtain approximate parameters for the potentials. For a simple exponential-repulsion, clocking experiments at different pump energies are enough to determine $V_1(t)$.

Finally, once $V_1(t)$ is determined, $V_1(R)$ can be obtained by integrating the equation of motion (Equation 12). Note that one need not assume a functional form for the PES, since the procedure can be numerically implemented, point by point, as shown schematically in Figure 5 with a VRT curve. Once $V_1(t)$ and $V_1(R)$ are determined, $V_2(R)$ can also be determined. Direct integration has already been shown to successfully invert different kinds of PES functions, including bound wells.[3,11] From the fits of the experimental ICN data on Figure 12, we can calculate the long range potential as a function of time and internuclear separation. Figure 15 shows the locations in $V_1(t)$ and $V_1(R)$ that were probed in these FTS experiments.

This inversion procedure is a powerful technique to invert FTS data into potential energy surfaces. By combining data sets obtained for different pump and probe wavelengths, both $V_1(R)$ and $V_2(R)$ potential energy surfaces have been obtained by our group.

4. Comparing classical and quantum mechanics

We now compare the quantum mechanical description proposed by Williams and Imre[5] to the results of this simple classical model. In their calculation, as in our classical model, only change along the internuclear coordinate was considered. The calculation includes a time dependent treatment of the two laser fields and the three electronic states. The pump and probe processes were simulated as curve crossings with a time dependent coupling between the two curves. Tuning the wavelengths is translated into changing the R position at which the two curves cross. The intensity of the laser has the effect of increasing the coupling between the curves.

Using the language of quantum mechanics, the pump process allows the ground state wave function to cross to the repulsive V_1 potential curve. On the repulsive potential, the wave packet, composed of a superposition of continuum wave functions, spreads and translates as a function of time. Figure 16 shows the evolution of the wave packet on the first excited state for delta function pulses, as well as positions calculated for a classical particle sliding on the same potential. The position of the particles is calculated with the simple classical formula derived here for R(t) (see Equation 15). The agreement in position between the quantum and mechanical translation is expected for delta function pulses because of the initial localization of the wave packet.

Figure 17 shows the evolution of wave packets prepared with 40 and 125 fs pump pulses (Figures 10 and 12 in Ref. 5); again the positions of a classical particle agree with the wave packet peak positions. Notice, however, that within the duration of the pump pulse ± 20 and ± 62.5 fs respectively, the wave packet shape does not look like a Gaussian and its average position cannot be clearly defined at early times. The dynamics occurring at these early times remain undetermined.

The results for the quantum mechanical model and the classical model are compared in Figure 18 for a series of hypothetical FTS experiments. The transients are taken for the same potential energy curves in Table 1 for V_2(not flat) and the same pump and probe wavelengths and pulse widths. As can be seen there is very good agreement except for the asymptote level which is a factor of square root of two larger for the quantum-mechanical simulations. The quantum-mechanical simulations are a plot of the population on V_2 and not the expected signal intensity. The multiplication factor arises from the multiplication of the wavefunction by the electric field of the laser and not the intensity (*i.e.* $I = E(t)^2$).[8]

The simple classical model described here has been found to fit the experimental data and to match the full quantum mechanical simulations.[6,8] In terms of computation times, it takes approximately a thousand times longer to perform a full quantum mechanical trajectory than it is to obtain a classical one. This additional advantage makes this simple classical model ideal for studying FTS transients.

5. Dynamics vs. pulse duration

With the goal of studying reaction dynamics in real-time, the inevitable questions are: how short should the pulses be, and, can the pulses be too short. The answer to the first question is simple, one could use picosecond or even nanosecond pulses, as long as two conditions are met. First, the pulses must be transform limited. Second, one must be able to determine the zero-of-time and any measured event with femtosecond time resolution. The only drawback to the use of long pulses is that they overlap a narrower section of the potential energy surface. The time spent by the intermediates on that fraction of the surface is very small compared to the duration of the pulse, therefore, the intensity is very small.

To answer whether the pulses can be to short, one must consider their bandwidth. A pulse with a bandwidth that spans the whole potential energy surface would measure only an instantaneous response. In the case of a bound potential, the coherent motion of the wavepacket prepared by the pump pulse causes a coherent beat pattern that can be analyzed by Fourier transformation to get the potential energy surface. For bound potentials, however, the pulses must have a sufficient bandwidth to span, at least, over two vibrational levels.

The fact that the bandwidth of the pulses is as important as the pulse duration is a manifestation of the uncertainty principle. It is interesting to plot a potential energy surface in the same graph as a pulse bandwidth in order to better understand the usefulness of short pulses. In Figure 19 the $\gamma/2$ for transform limited Gaussian pulses is plotted (frequency vs. time) together with the potential energy surface $\Delta V(t)$ for the two sets of potentials in Table 1. The top graph corresponds to V_2(flat). It is clear that the slower dynamics allow for longer pulses (see top graph in Figure 19). From these graphs, one can chose the ideal pulse durations by selecting a pulse whose $\gamma/2$ is centered around the region of the potential which is of interest. For the early parts of the potential, it is quite important to have short pulses. For the long range, the pulses can be quite long.

The fact that the pulse duration and bandwidth are intimately related to the measurements has been a source of controversy. If one obtains different $\tau_{1/2}$ values for a given reaction, depending on the pulses used, then what is the significance of these measurements. The answer is simple, each measurement determines the potential energy surface (see Equation 22) which, ultimately, is what one is after.

6. Discussion and extensions of the classical model

For the case of bound potentials such as are found in the dissociation of NaI[9] or in molecular iodine,[10] this technique is able to obtain the turning points on the potential at a given energy, but is not able to recover the potentials from direct inversion as proposed in Section 3 due to the quantum mechanical properties of resonances. In fact, a new classical treatment by Bernstein and Zewail has proposed the extraction of the turning points by analyzing only the period between peaks and ignoring the overall shape of the transient.[11]

In addition to the problems that arise in a one dimensional system, two and three dimensional systems are much more complicated and need to be simplified in order to use the models mentioned here. The concepts discused here have been used to yield qualitative and semi-quantitative information for more complicated systems. In terms of time dependent alignment during dissociation, FTS transients can be obtained free of alignment effects by making use of the polarization properties of the pump and probe lasers. For problems involving more than one dimension such as the dissociation of HgI_2, this treatment is able to predict the outcome of clocking experiments after subtraction of the energy left in the fragments due to vibration and rotation. The calculations can be performed for any of the three dissociation channels of the reaction.[12]

Overall, this simple classical treatment has been very useful in providing insight into the novel technique of real-time observation of transition states. Its simplicity clarifies the effect of tuning the pump and probe pulses, as well as their relation to the time delay between them. It also allows a physical picture of the potential energy surface in terms of the observed FTS transients. Understanding the underlying approximations of the technique allows one to predict the cases for which the model is quantitatively correct. Being a classical model, it is not expected to be correct when quantum mechanical effects are important, but this was never the intention. The reader will notice in the chapter containing the experimental applications of the FTS technique that most of the initial insight on the dynamics comes from the application of this simple model.

References

1. M. Dantus, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 87, 2395 (1987).
2. R. Bersohn and A. H. Zewail, Ber. Bunsenges. Phys. Chem., 92, 373, (1988).
3. R. B. Bernstein and A. H. Zewail, J. Chem. Phys. 90, 829 (1989).
4. M. Dantus, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 89, 6128 (1988).
5. S. O. Williams and D. G. Imre, J. Phys. Chem., 92, 6648, (1988).
6. L. E. Fried and S. Mukamel, to be published.
7. I. Benjamin and K. R. Wilson, J. Chem. Phys., 90, 4176, (1989); J. L. Krause, M. Shapiro and R. Bersohn, to be published.
8. Work in progress in our group.
9. T. S. Rose and M. J. Rosker and A. H. Zewail, J. Chem. Phys., 88, 6672, (1988); T. S. Rose and M. J. Rosker and A. H. Zewail, Chem. Phys. Lett., 146, 175, (1988).
10. R. M. Bowman, M. Dantus and A. H. Zewail, Chem. Phys. Lett. 161, 297 (1989); M. Dantus, R. M. Bowman and A. H. Zewail, Nature 343, 737 (1990); M. Gruebele, G. Roberts, M. Dantus, R. M. Bowman and A. H. Zewail, Chem. Phys. Lett. 166, 459 (1990).
11. R. B. Bernstein and A. H. Zewail, Chem. Phys. Lett., 170, 321 (1990).
12. R. M. Bowman, M. Dantus and A. H. Zewail, Chem. Phys. Lett., 156, 131, (1989); M. Dantus, R. M. Bowman, M. Gruebele and A. H. Zewail, J. Chem. Phys., 91, 7437, (1989).

Table I: Parameters of the potential energy surfaces for the simulations.

For V_2 flat:	For V_2 not flat:‡
$V_1 = 6{,}000$ cm^{-1}	$V_1 = 7{,}000$ cm^{-1}
$L_1 = 0.8$ Å	$L_1 = 0.33$ Å
$C_1 = 25, 679$ cm^{-1}	$C_1 = 25, 679$ cm^{-1}
$V_2 = 0$ cm^{-1}	$V_2 = 6{,}584$ cm^{-1}
$L_2 = $ N/A	$L_2 = 0.35$ Å
$C_2 = 51{,}423$ cm^{-1}	$C_2 = 51{,}423$ cm^{-1}

for $V_i(R) = V_i(R_e) \exp\{- (R-R_e)/L_i\} + C_i$,

‡ These parameters correspond to those of Reference 5 when the excitation wavelengths is 306 nm.

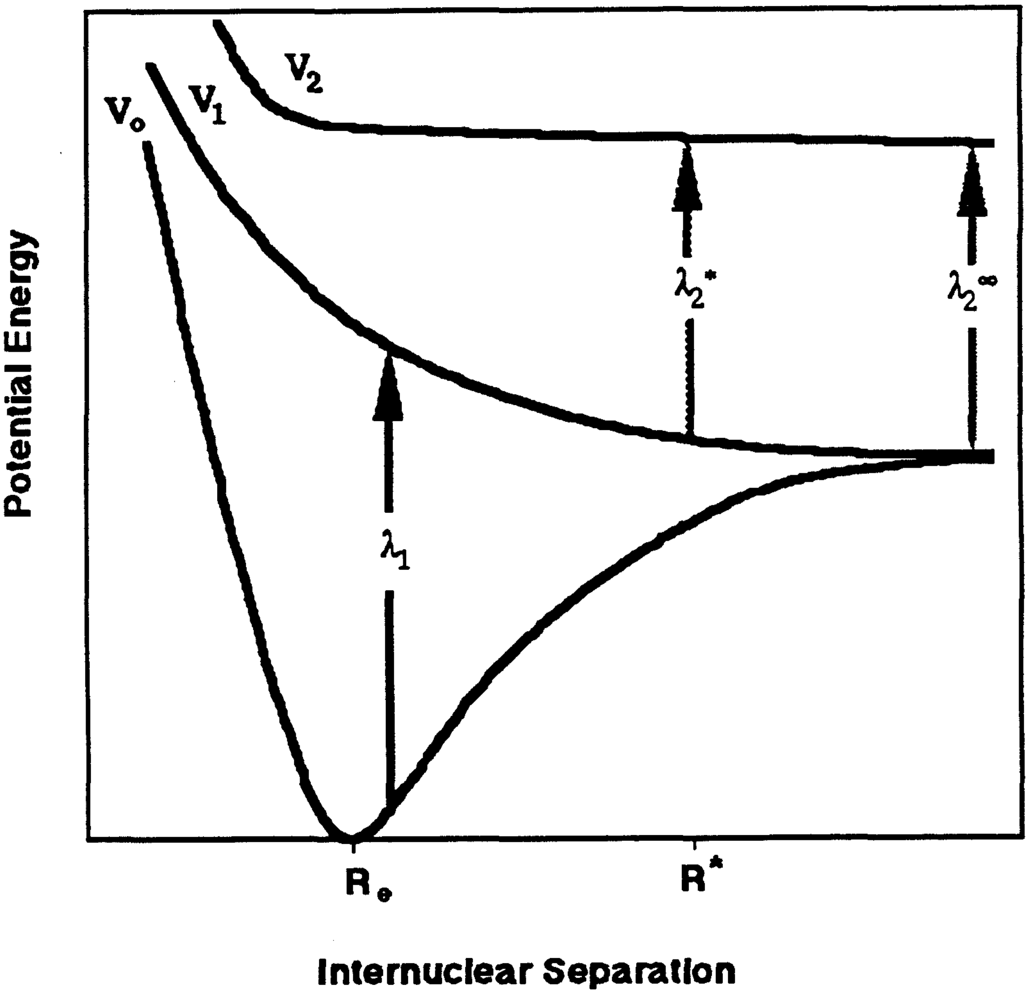


Fig. 1 Schematic drawing showing the concept of the FTS technique. The potential energy surfaces are representative of an idealized diatomic or quasi-diatomic molecule. The reaction is initiated by the pump pulse with wavelength λ_1 which promotes the molecules from the ground state V_0 to the repulsive surface V_1. Once in the repulsive state, the evolution towards free fragments along the repulsive surface is probed with wavelengths $\lambda_2{}^*$. To measure when the reaction is complete the probe wavelength is tuned to $\lambda_2{}^\infty$. The probe wavelength is tuned to frequencies that match the energy difference between V_1 and V_2 at a given internuclear distance R*.

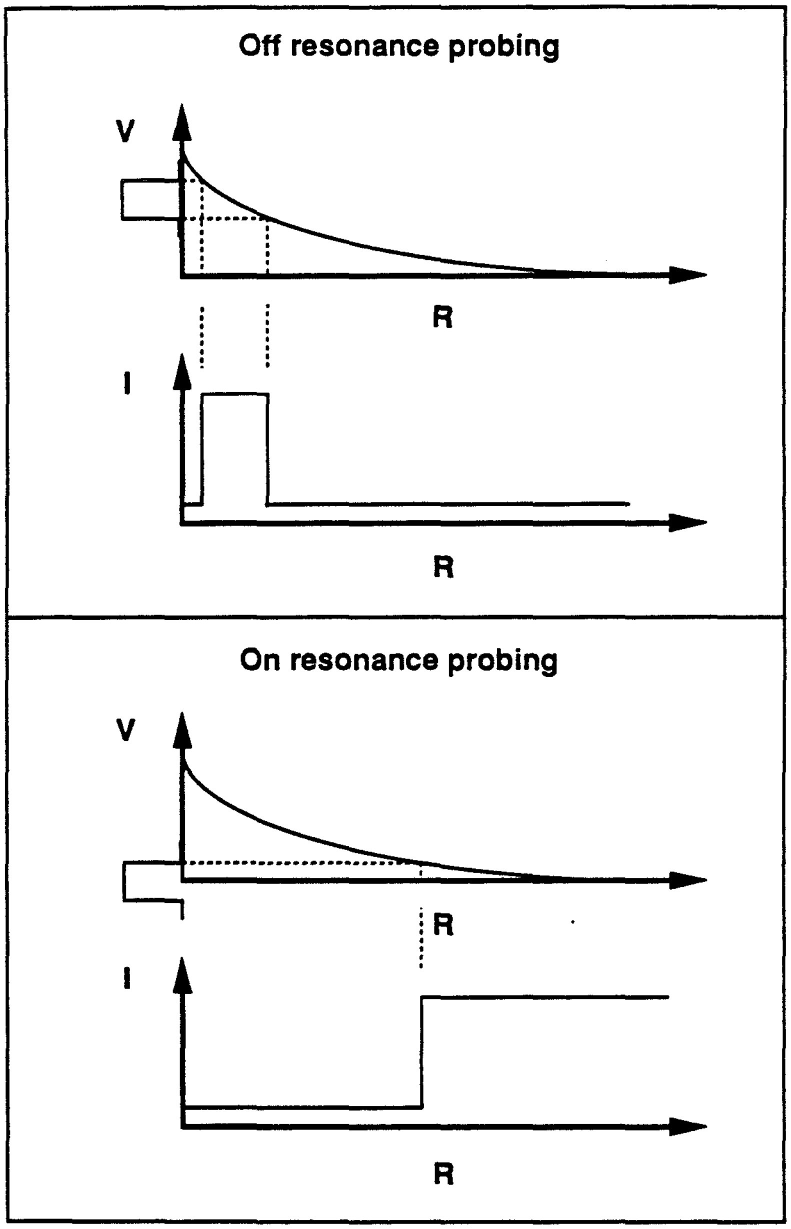


Fig. 2 The probe laser can be considered as opening a widow on the potential. This schematic shows the effect of a square-shaped window on a repulsive potential V as a function of internuclear distance R. The lower plot shows the relative intensity I (or absorption) as a function of R caused by the probe window. The concept is shown for off-resonance probing (top) and on-resonance probing (bottom).

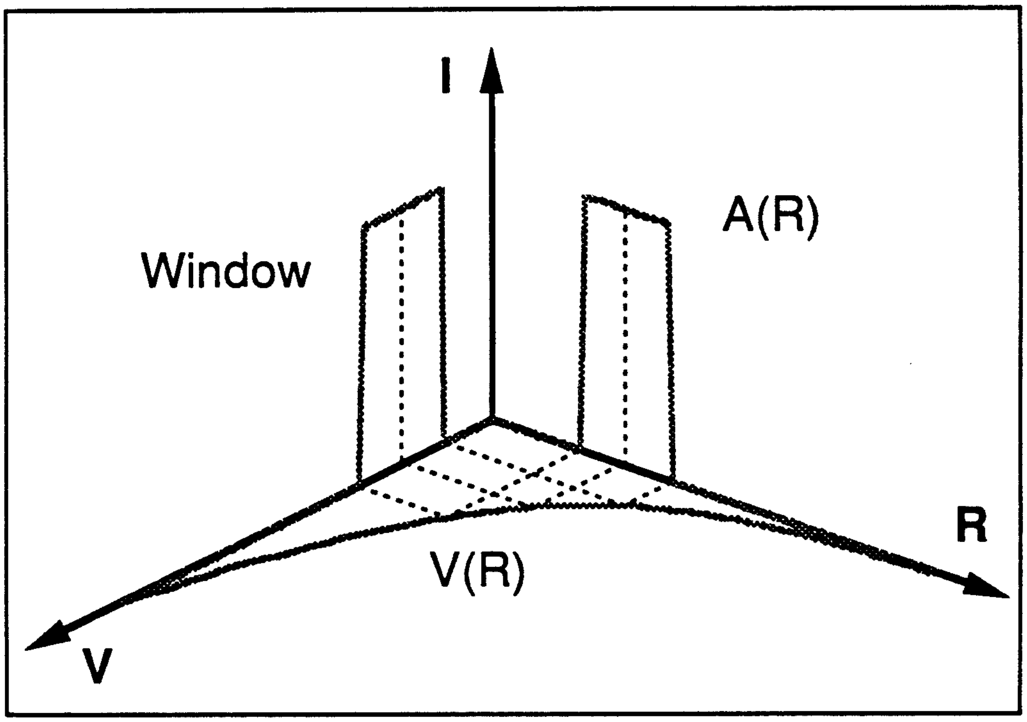


Fig. 3 Three dimensional schematic showing the relation between the probe laser (the window), the potential energy surface V(R) and the absorption as a function of internuclear distance A(R).

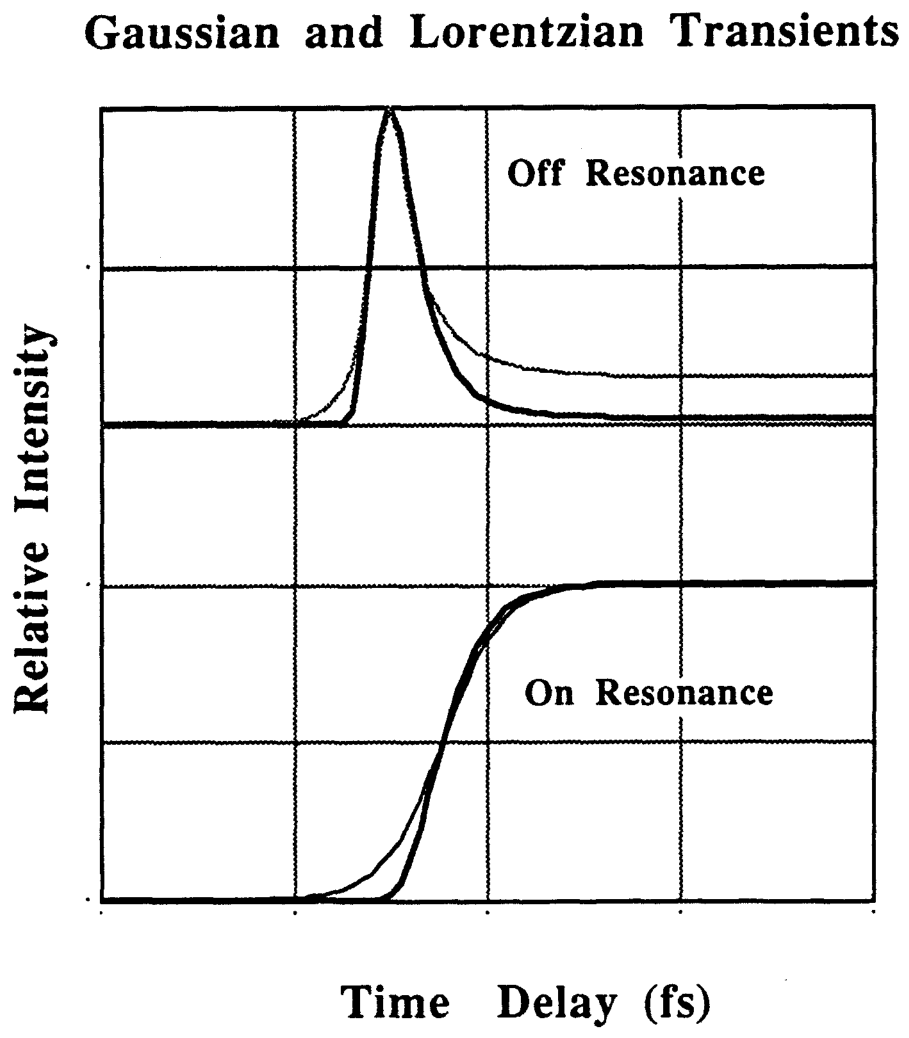


Fig. 4 Calculated transients for Gaussian and Lorentzian pulse shapes. Note that the Lorentzian pulse shape introduces much more intensity in the wings.

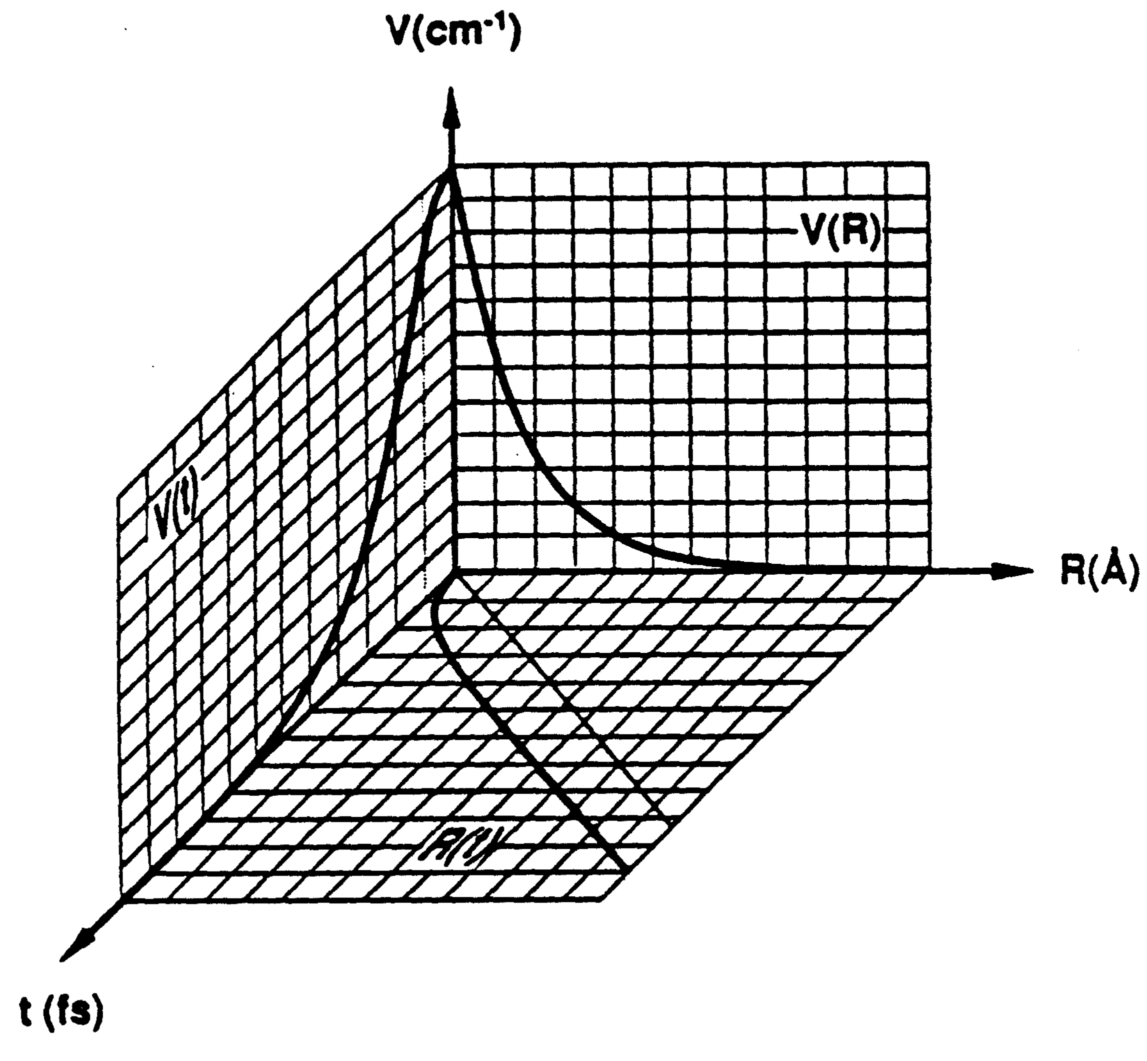


Fig. 5 Three dimensional schematic showing the relation between the time (t = 25 fs/div) the potential energy (V = 500 cm^{-1}/div) and the internuclear distance (R = 0.5 Å/div) for a dissociation reaction with an exponential repulsion. Note that the recoil velocity, given by the instantaneous slope of R(t) vs. t, rapidly achieves the terminal value. The straight line corresponds to the terminal velocity of the reaction.

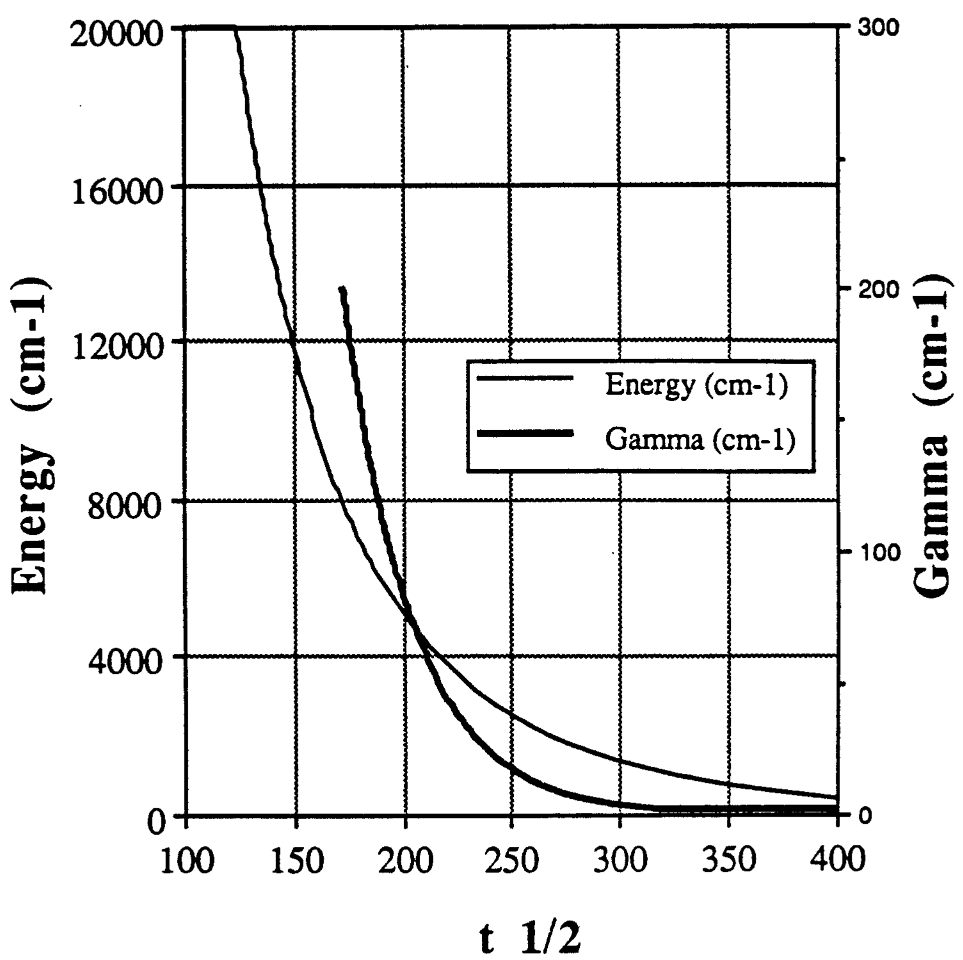
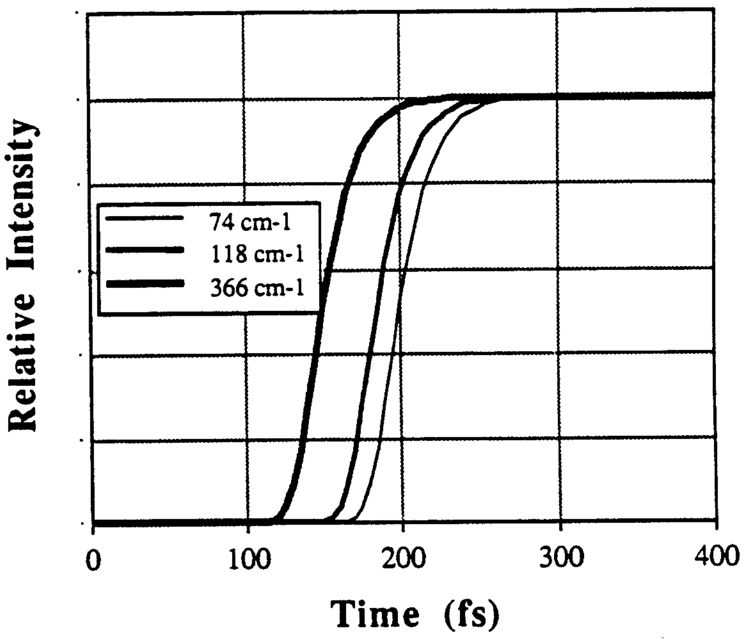


Fig. 6 Plot showing the variation of $\tau_{1/2}$ vs. the total energy for the reaction (Gamma fixed at 118 cm^{-1}) and the probe spectral width Gamma (Energy fixed at 6000cm^{-1}). As expected increasing the total energy makes the reaction faster.



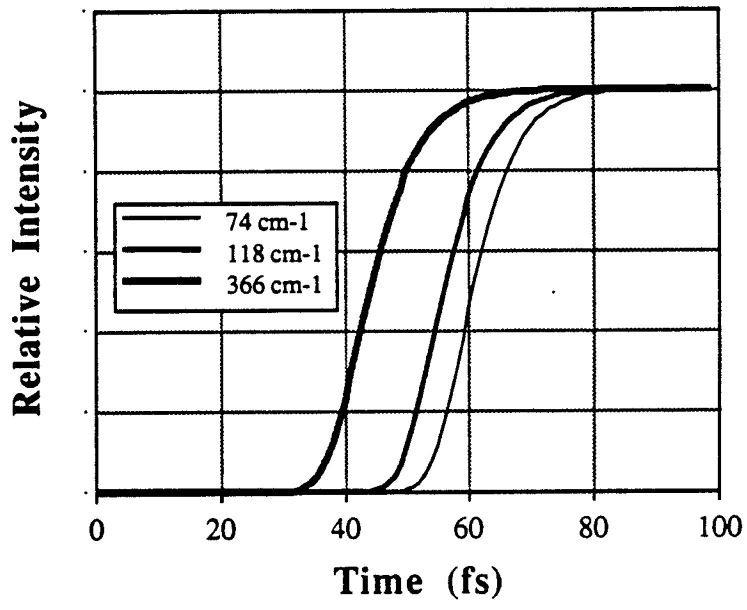


Fig. 7 Calculated on-resonance transients for potentials $V_2(R)$ flat (top) and $V_2(R)$ not flat (bottom). The parameters are given in Table 1. The transients are given for different spectral pulse widths.

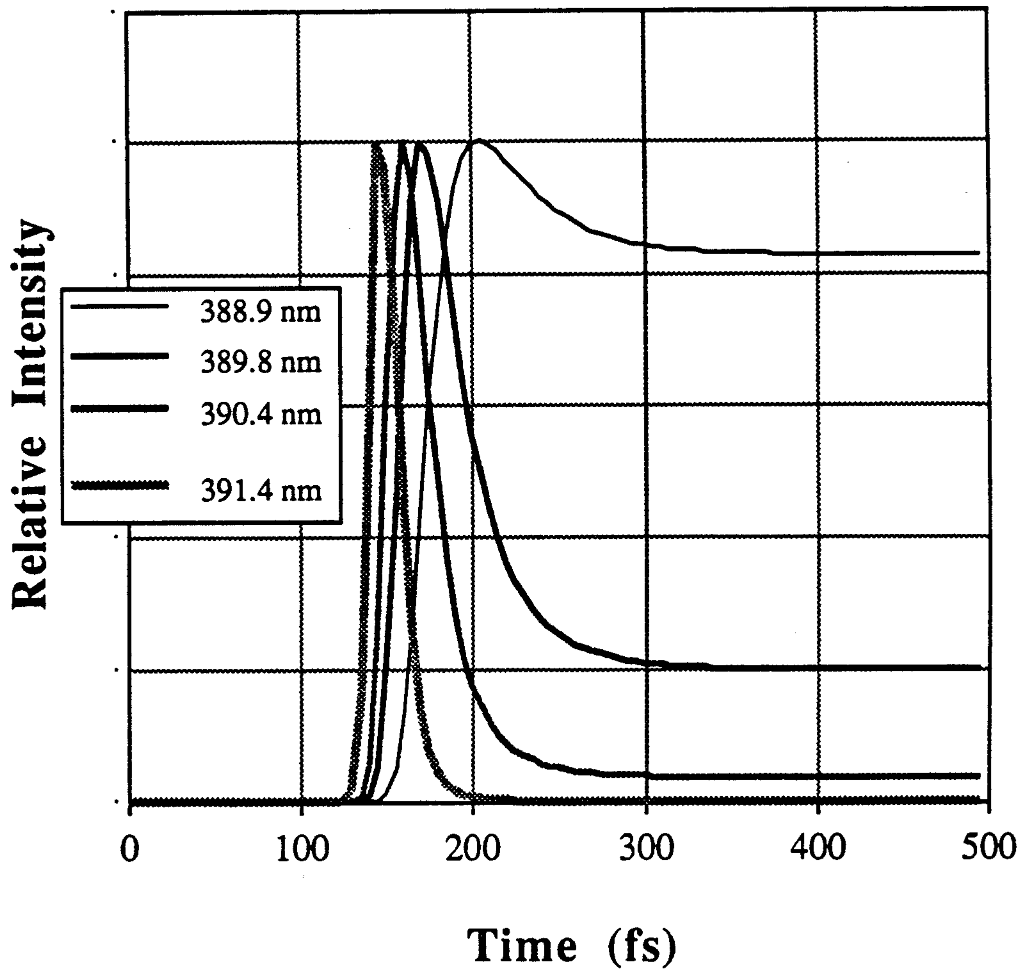


Fig. 8 Calculated FTS transients as a function of the probing wavelength. The parameters of the PES are those of V_2 flat in Table 1. The pulse spectral width is 118 cm^{-1}.

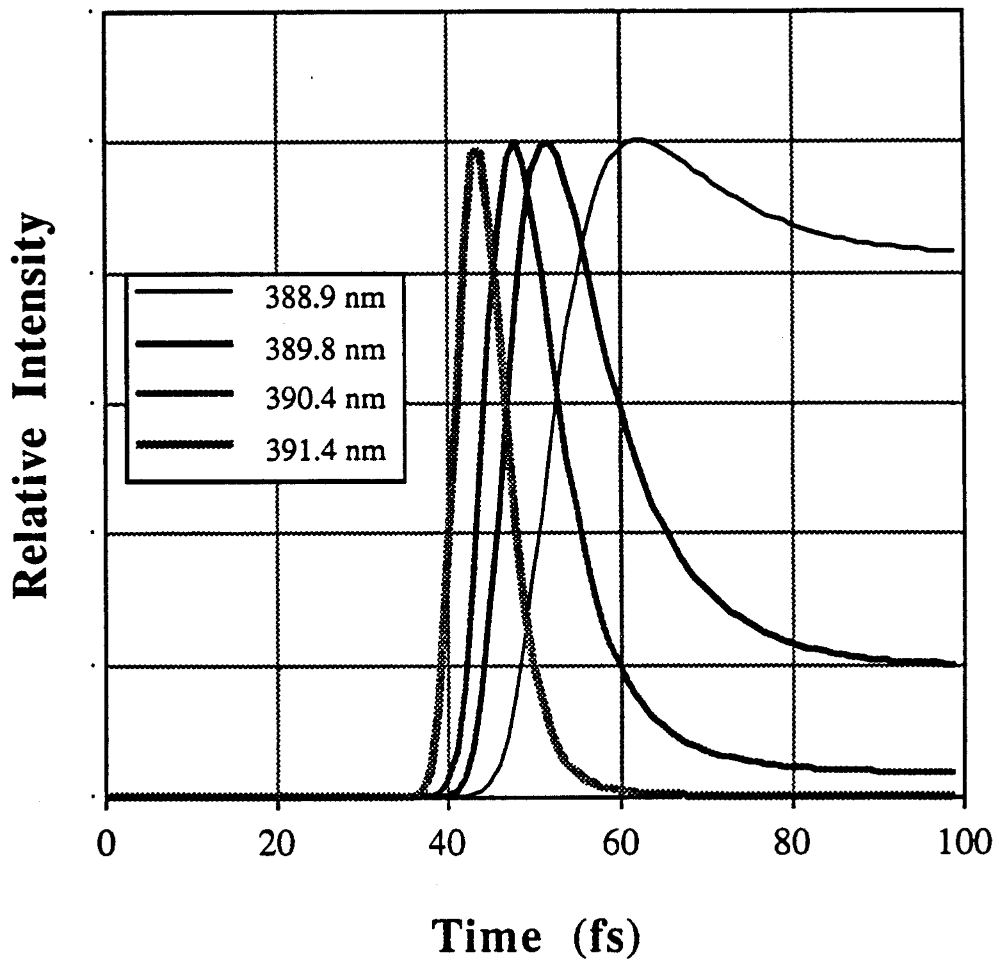
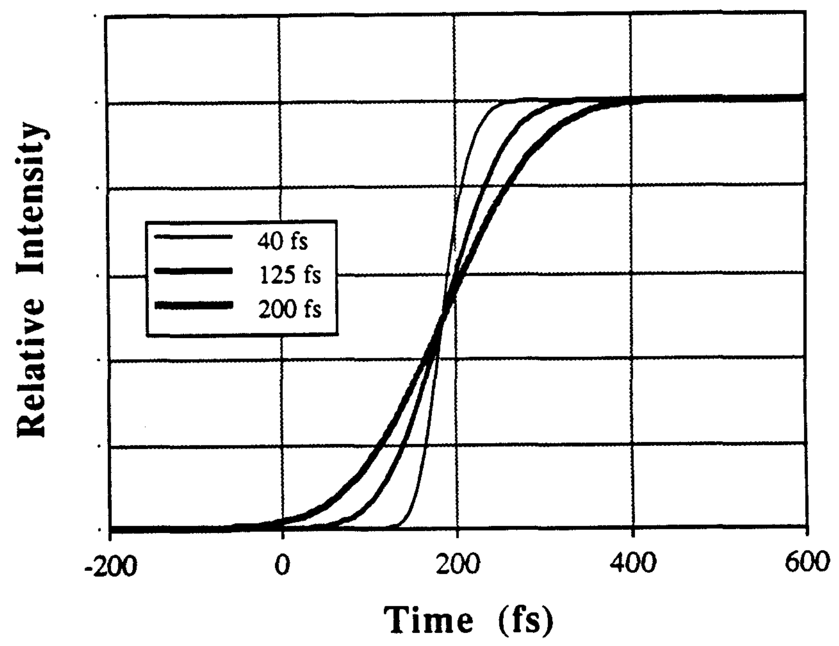


Fig. 9 Calculated FTS transients as a function of the probing wavelength. The parameters of the PES are those of V_2 not flat in Table 1. The pulse spectral width is 118 cm^{-1}.



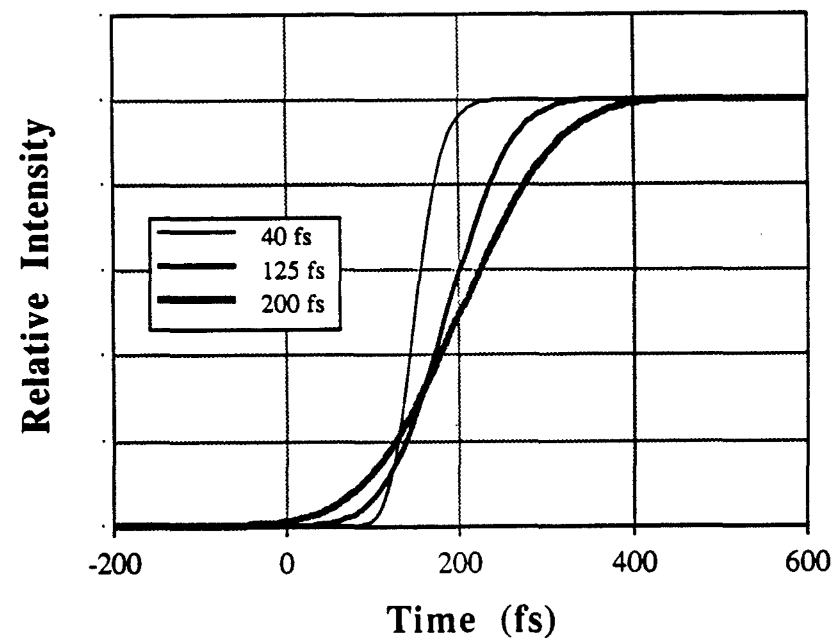


Fig. 10 Calculated FTS transients showing the effect of convolution for different pulse widths. The top graph shows the effect of convolution with different pulses without taking into account that different pulse durations have different spectral widths. The bottom graphs corresponds to the convolution effect taking into account the temporal and the frequency characteristics of transform limited Gaussian pulses.

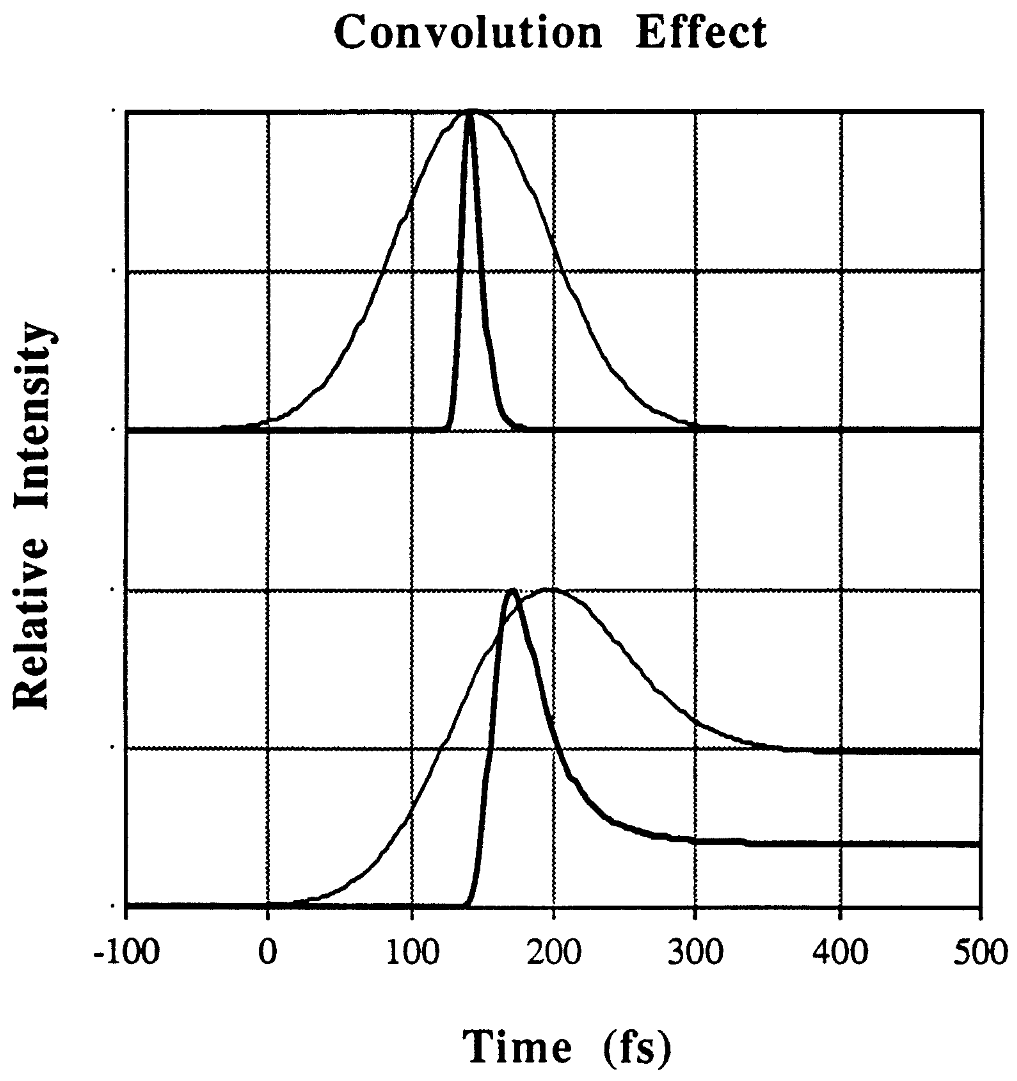


Fig. 11 Calculated effect of convolution for off resonance transients. The transients are calculated for transform limited 125 fs Gaussian pulses. Note that for the far off resonance case (top) the convolution does not shift the peak position or the asymptotic level. For near-resonant transients (bottom) the convolution process shifts the peak and increases the asymptotic value.

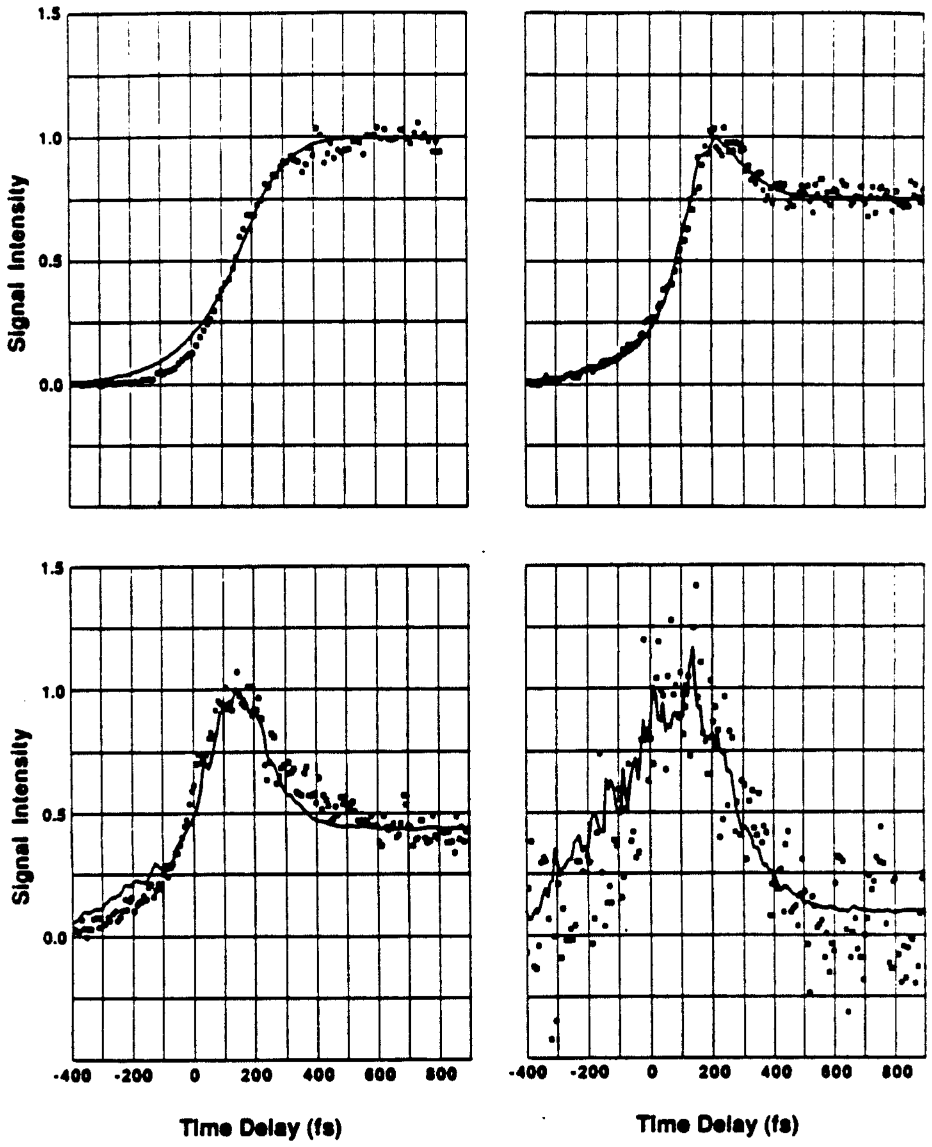


Fig. 12 Typical ICN transients as a function of probe wavelength. The probe wavelengths were (clockwise) 388.9, 389.8, 390.4 and 391.4 nm. The solid lines were generated by fitting with the classical model here discused and convolved with the measured experimental response function.

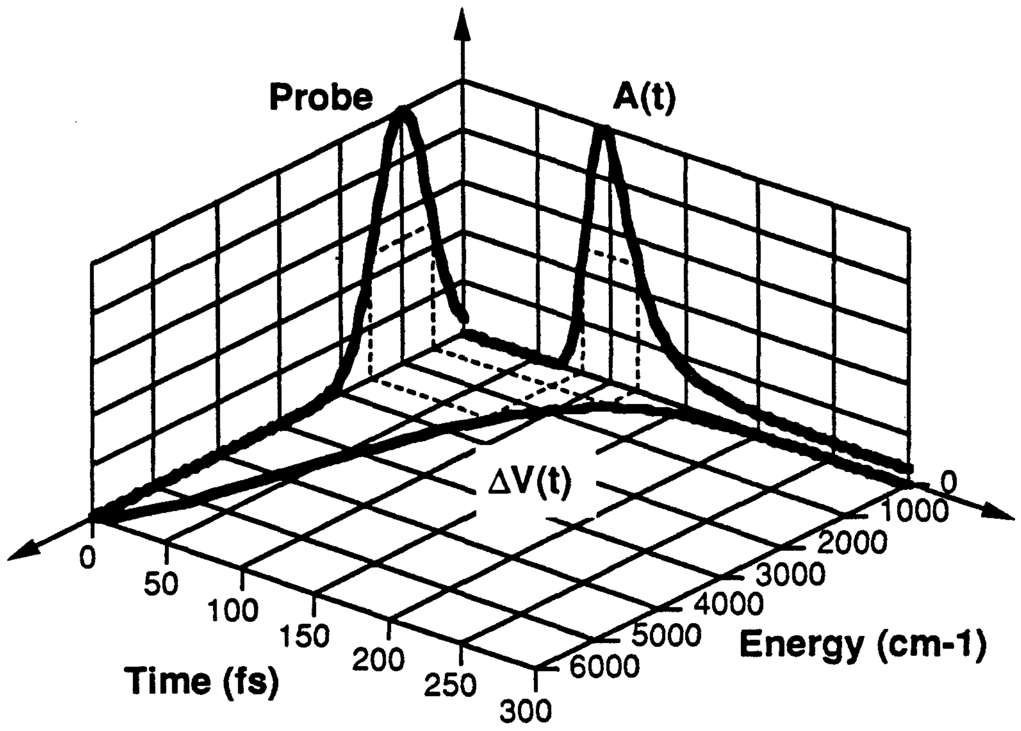


Fig. 13 Three dimensional schematic showing the effect of a Gaussian probe laser opening a window on the potential energy surface. The curve A(t) corresponds to the measured FTS transient. The dashed lines show the direct relation between the shape of the probe and the shape of the transient. This relation is used for the inversion technique (see text).

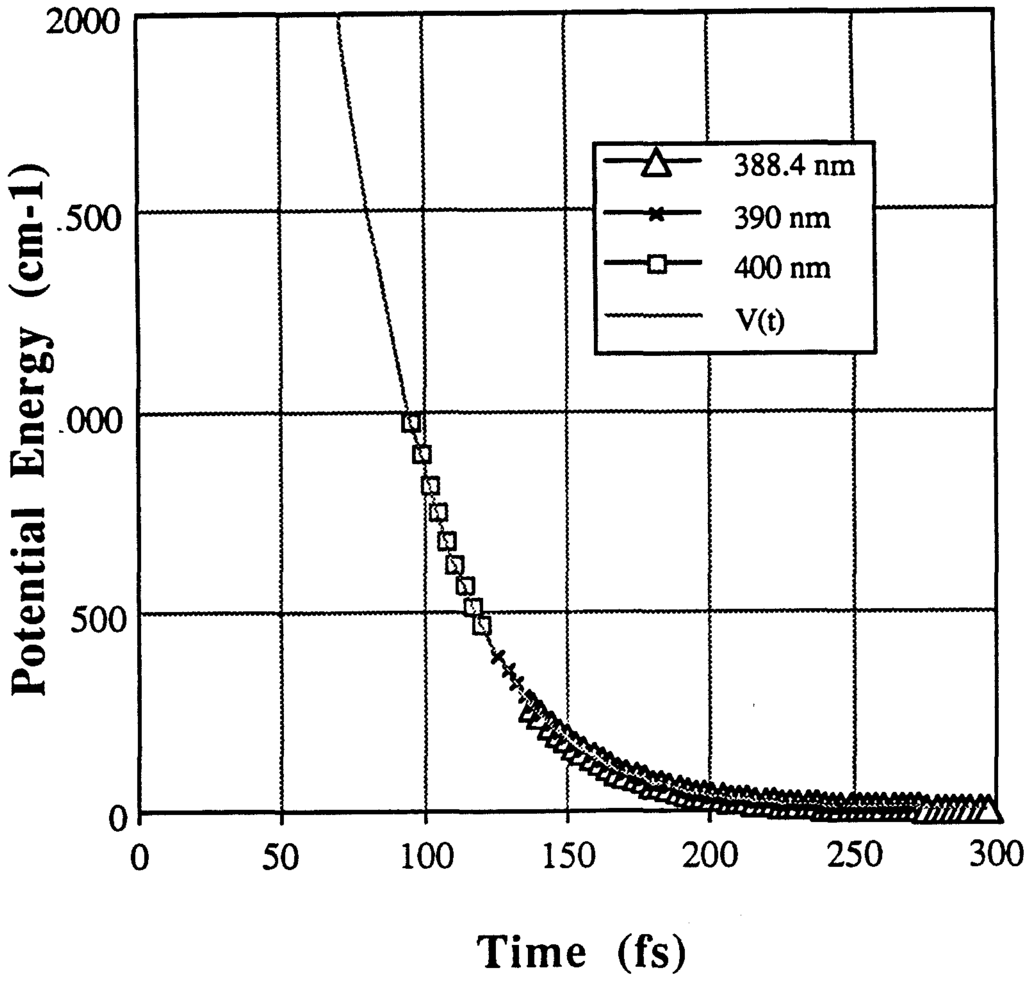


Fig. 14 Results from the inversion of calculated FTS transients at different probing wavelengths. Note that at each wavelength the transients are sensitive to different regions of the potential.

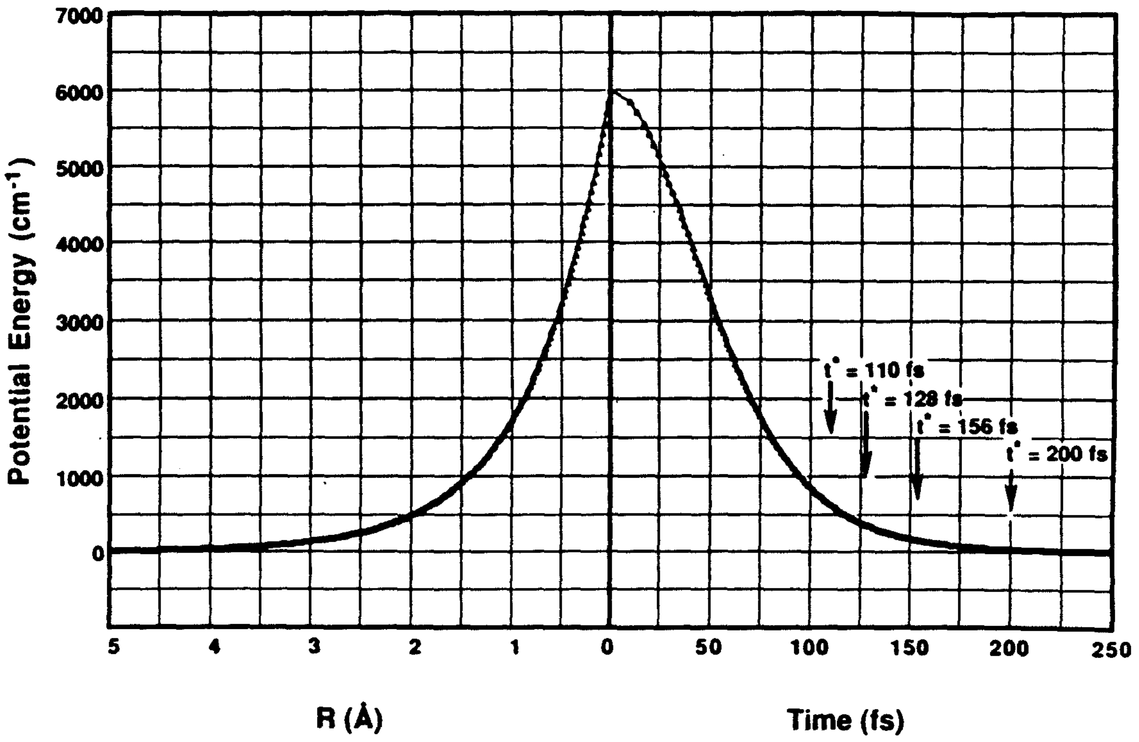


Fig. 15 Plot of V(R) and V(t) showing the regions or R and time probed by the FTS experiments on the dissociation reaction of ICN.

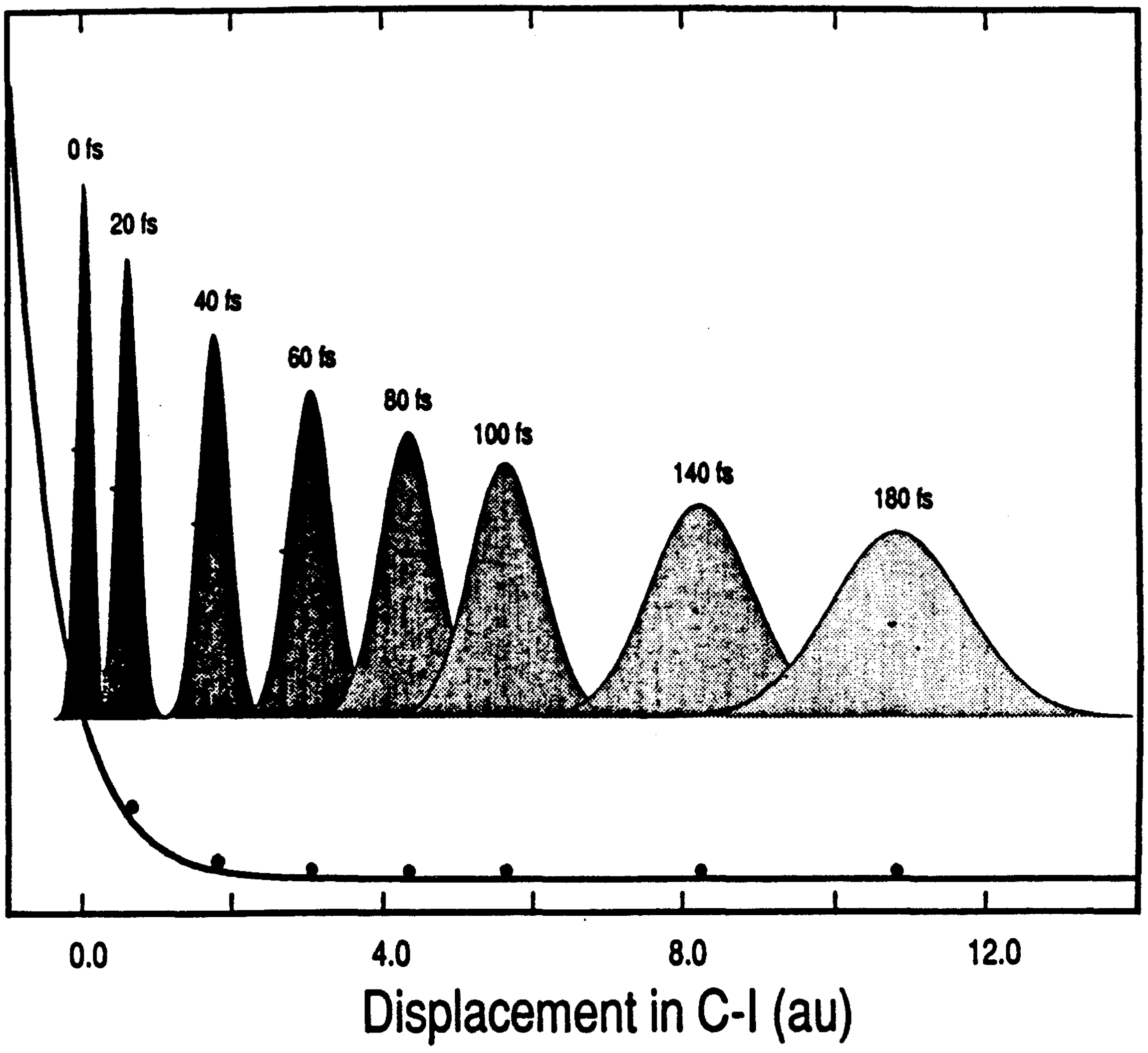
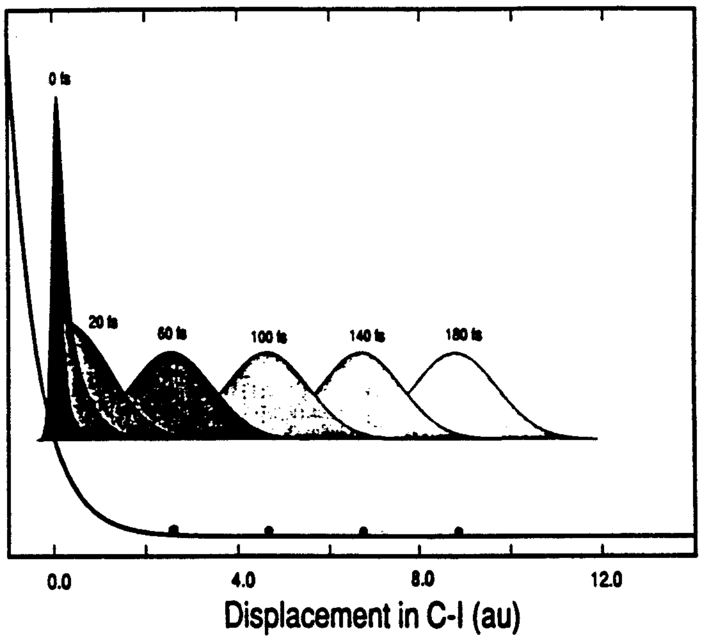


Fig. 16 Figure taken from Reference 5 showing the results of the propagation of a wave packet generated with delta function pulses. The black dots on the potential are the calculated position of a classical particle in a dissociation trajectory. Both calculations were made with the same PES parameters. Note that the position of the particle corresponds to the center of the wave packets.



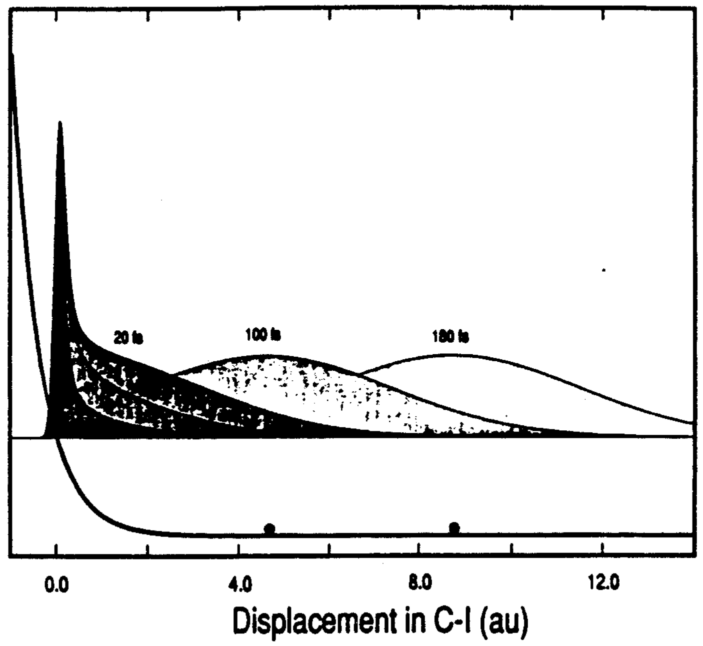


Fig. 17 Figures taken from Reference 5 showing the results of the propagation of a wave packet generated with 40 fs pulses (top) and 125 fs pulses (bottom). The black dots on the potentials are the calculated position of a classical particle in a dissociation trajectory. Both classical and quantum mechanical calculations were made with the same PES parameters. Note that the position of the particle corresponds to the center of the wave packets, however, the classical calculation neglects the spread of the wave packet.

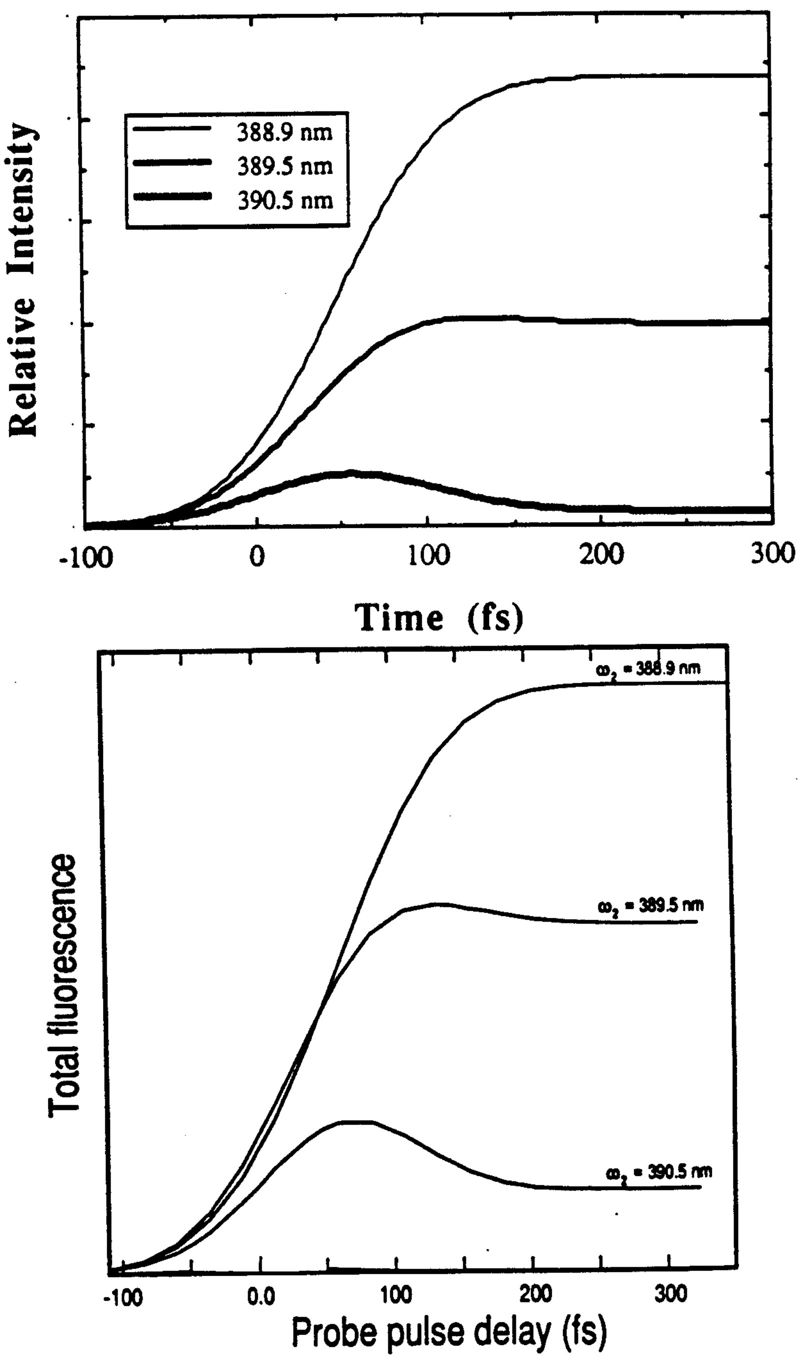
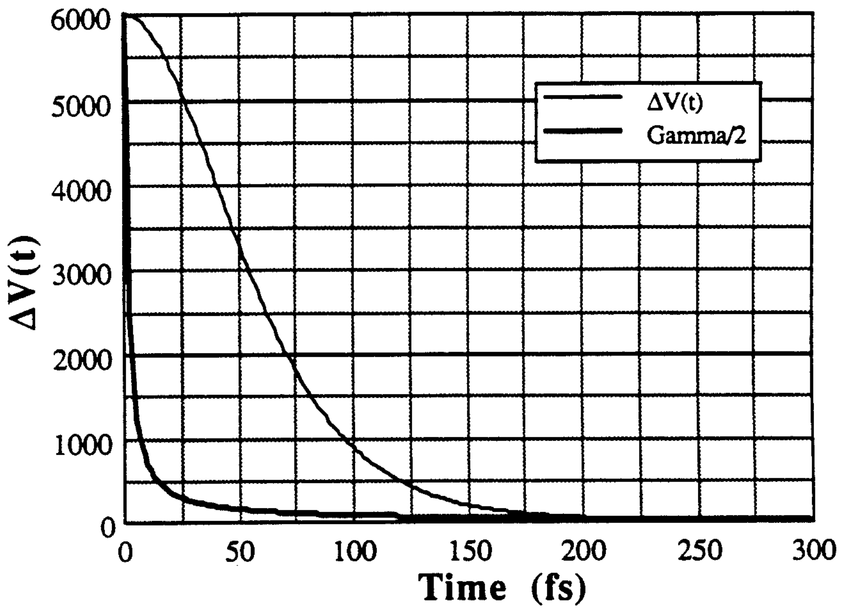


Fig. 18 Comparison between the classical and the quantum mechanical calculations. (Top) Calculated FTS transients for transform limited 125 fs Gaussian pulses for three different wavelengths using the parameters for $V_2(R)$ not flat. (Bottom) transients from Reference 5, calculated with the quantum mechanical model.



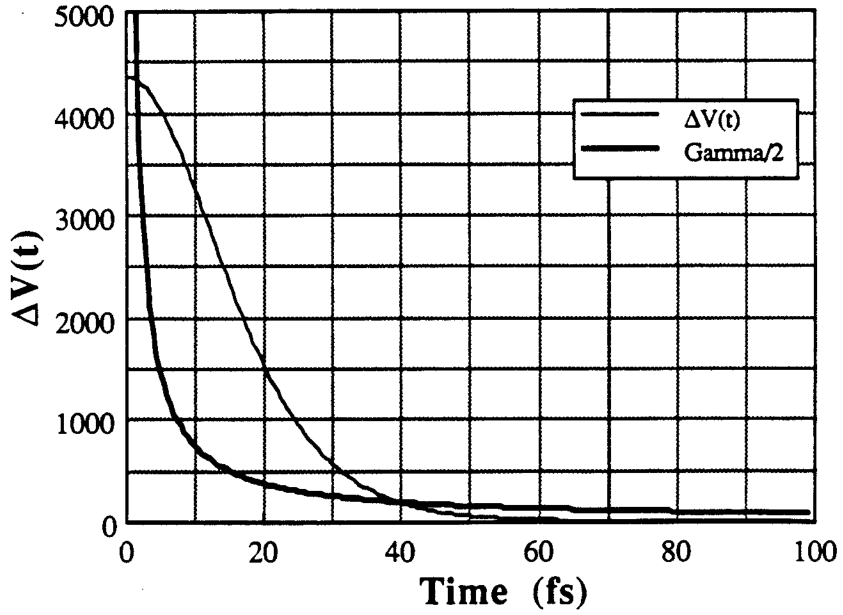


Fig. 19 Plots showing the potential energy difference, $\Delta V(t) = V_2 - V_1$, as a function of time for both flat (top) and not flat (bottom) PES parameters in Table 1. Included in the graphs is the dependence of spectral pulse width of transform limited Gaussian pulses.

Chapter III

Experimental

The experimental challenge set forth in 1986 by Professor A. H. Zewail was to design a laboratory capable of measuring in real-time the dynamics of elementary chemical reactions in the gas phase. The time resolution needed for these kinds of measurements was 10^{-14} seconds. Techniques for the generation of ultrafast laser pulses had been demonstrated by groups in AT&T Bell laboratories, but the long term stability of those laser systems and their applications to the study of chemical reaction dynamics was yet to be explored. The premise was that time resolved experiments would reveal dynamics that could not be inferred by time integrated techniques. Picosecond lasers had already been used in our group to study state-to-state rates of chemical reactions, but in many cases, shorter time resolution was needed. This laboratory would allow the possibility of observing in real-time the passage of reactants to products in a chemical reaction with enough resolution to record the dynamics of the species in transition between reactants and products, here referred to as "the transition states."

The design and construction of the Caltech Femtosecond Laser Facility was dictated by the goal of time-resolving the dynamics of chemical reactions. The description of the laboratory is organized as follows. (1) A brief description of the design of the laboratory is given in order to emphasize the sensitivity of a femtosecond laser system to the environment. The laser system, which was inspired by descriptions available in the scientific literature, is the result of hours of theoretical and experimental effort by Dr. Mark J. Rosker, Marcos Dantus and later improvements by Dr. Todd S. Rose and Dr. Robert M. Bowman. The current design is described here in detail. (2) A brief review of the available techniques for the generation of femtosecond pulses and mode-locking follows, including a detailed section on our generation scheme. (3) A detailed description on amplification of femtosecond pulses is also included.

In addition, (4) pulse compression techniques used after the amplification process to shorten the pulses are discussed. (5) The characterization of femtosecond pulses in time and frequency, a routine experimental procedure, is also included. (6) Non-linear generation techniques used to prepare femtosecond pulses at different wavelengths from the infrared to the ultraviolet are presented. (7) Experimental considerations for the study of chemical substances using the FTS technique are discussed, and the data acquisition procedure is described.

1. The Caltech femtosecond laser facility

The demands of a femtosecond laser system are much more stringent than those of other laser systems. The time resolution of the apparatus depends on the duration of the pulses and the precision to which one can control the delay between two pulses. Consider that we have generated a pair of ultrafast laser pulses and are measuring the time difference between them. The resolution of this measurement will be limited to the accuracy with which we can control the path length that each pulse travels. If we were able to generate two infinitely short pulses but could only control the difference in path length between them by one thousand of an inch, our time resolution would be 85 fs. This time resolution is determined by calculating the time it takes light to travel the specified distance. In our laboratory, the desired pulse duration is 40 - 50 fs with a resolution of ± 5 fs, therefore distances must be determined with an accuracy of 0.06 thousands of an inch, or equivalently, 1.5 microns. The distance scale is so small that a 50 fs pulse with a central wavelength of 620 nm consists of only 24 wave cycles and the 5 fs resolution is slightly over two wave lengths. This kind of precision requires special vibration and temperature control which is described here.

The facility was constructed in the sub-basement since it is the most quiet part of the building structure. This was confirmed by seismic measurements. The inherent structural motions of the building still required an optical bench with vibrational isolation to mount the laser setup. With a set of four vibrational isolators, the passive vibrational isolation system reduces considerably the vibrations of the floor. Essentially, the optical bench is floating on high pressure nitrogen. The performance of the optical bench was tested in situ. The results are displayed in Figure 1 showing the suppression of vibrational frequencies of the bench when it is floating.

Wind drafts were reduced by an acrylic box placed over the entire sensitive portion of the optical setup. The box keeps drafts off the laser system with a noticeable increase of stability. The need for cleanliness of a laser has been usually underestimated. The total number of optics in the setup is estimated in the low hundreds. Most of the optics have single surface optical coatings which are easily damaged by cleaning; other optics such as first-surface silver mirrors cannot be cleaned at all. It is very important to keep dust away from all optics. The small diameter of the laser beams make them vulnerable to dust particles that may block a considerable percentage of the beam. Since the laser system is very sensitive to intra cavity intensity, a dusty environment could reduce the stability considerably. The air conditioner was designed to filter 95% of the particles which are 2 microns in diameter. All larger particles are completely removed, making the laboratory a relatively clean room.

Looking back on three years of experiments on the Femtosecond Laser Facility, it is clear that what seemed at times as exaggerated demands on the laboratory is what has kept the laser system working with outstanding stability and reliability. Every instrument and optic, including the large frame lasers, has outlasted even the most optimistic expectation of their manufacturers. The argon ion laser tube lasted three times the expected lifetime and our uncoated non-linear crystals have lasted now six times their expected lifetime, this due to the reduced humidity in the laboratory. Overall the design of this facility has confirmed that careful planning is an important investment, and has made this laboratory a model for new femtosecond laboratories in our group.

2. Generation of femtosecond pulses

Since the first report on the mode-locking of the ruby laser in 1965[1] there has been a quest for generating the shortest optical pulses. By 1966, picosecond pulses were generated with a mode-locked Nd:glass laser.[2] The progress towards shorter pulses was based on breakthroughs such as active mode-locking,[2] flashlamp dye lasers,[3] active mode-locking of dye lasers,[4] colliding pulse mode-locking,[5] and pulse compression.[6] By 1986, 20 years after the first picosecond pulses were generated, a 6 fs pulse was generated and measured by Fork et al.[7]. This four orders of magnitude reduction in pulsewidth does not mean that every two years we can expect a reduction by a factor of a third. Actually, in the last four years there has been no further pulse reduction.

Progress in the generation of femtosecond pulses has not stopped, but it is under consolidation. Many different mode-locking techniques are now available and the future seems to involve all solid-state components.[8] The generation of pulses shorter than 6 fs presents great challenges. Consider a 2 fs pulse with a bandwidth spanning the whole visible spectrum from 300 to 700 nm. Such a pulse, taking the central wavelength to be 500 nm, would consist of only one optical cycle. Further reductions in pulse width would require additional spectral components but the pulse will always be composed of a single wave cycle. Due to dispersion in the air, the generation and measurement would have to take place under vacuum, thus requiring an entirely different pulse generation scheme than is currently used. One possibility would be to prepare a linearly chirped pulse which is then compressed in a vacuum chamber; another would be to generate the pulse in a free electron laser facility.

The application of femtosecond lasers to physics, chemistry and biology is still in the first stages of development and pulse durations of 50 fs are powerful enough to explore dynamics that have never been observed. To generate these kinds of pulses there are two basic techniques, sinc-pumped laser systems and colliding-pulse mode-locking systems. In our laboratory, we have chosen the latter for reasons of stability, as will be discussed below.

For our laser system, the oscillator is a colliding pulse mode-locked laser (CPM).[5] The CPM is a rather simple arrangement of optics with no electronics that, when pumped with a continuous laser, generates pulses as short as ~ 40 fs. The description of the CPM is very important since it introduces some aspects of ultrafast phenomena which will become relevant in following chapters. Its description is divided as follows: (1) the ring dye laser, (2) the saturable absorber (introduction to passive mode-locking), (3) the prisms (introduction to group velocity dispersion) and (4) operation the CPM laser.

2.1 The ring dye laser

The skeleton of the CPM (see Figure 2) is a ring dye laser cavity consisting of seven mirrors. The design is a version inspired by a similar laser by Valdmanis et al.[9] The dye laser is simply an optical resonator that has, as a gain medium, an organic dye. This laser dye, Rhodamine tetrafluoroborate[10] (R6G), is pumped by a continuous laser, in this case the 514.5 nm line of an Innova100-20 argon ion laser, to a vibrational level in its first excited singlet state (S_1) (see the insert in Figure 3). The excited vibrational state decays to the bottom of S_1 due to inter- and intramolecular energy redistribution taking place in ~10^{-12} sec. Fluorescence from the bottom of S_1 to different rovibrational states in S_0 accounts for the broad bandwidth fluorescence of laser dyes. The typical pumping power is 3 - 5 watts. The beam is focussed to a spot size of ~ 10 μ diameter on the gain jet, resulting in ~6×10^6 watt/cm^2.

The dye laser cavity (see Figure 2) is made up of the gain jet (GJ) and mirrors A, B, C, G1 and G2. Mirrors G1 and G2 are spherical mirrors (R = 10 cm) and serve to collimate a portion of the fluorescence emitted by the gain medium, while mirrors A, B, and C form the ring in the shape of a right angle triangle. Mirror C (also called the output coupler) is 98% reflective, allowing 2% of the light to escape. The gain is a vertical flowing solution of R6G in ethylene glycol. The concentration is adjusted until the jet is 13% transmissive at the argon laser wavelength (514.5 nm). Operation of the laser with the five mirrors mentioned and the gain jet results in a continuous laser that will lase with a wavelength in the range of the R6G fluorescence spectrum (580 - 630 nm see figure 3) depending on the alignment of the laser. The output of this arrangement, when pumping with 4 w of 514.5 nm, is typically 200 mW per arm. It is important to notice that there is no restriction for the light to travel in any direction (clockwise or counterclockwise). The output is, therefore, composed of two beams. For reasons which will become apparent in Section (2.4), the beam used for experiments is the one belonging to photons travelling clockwise.

2.2 The saturable absorber and passive mode-locking

In this section, we discuss how to get pulses out of a continuously pumped laser cavity. As mentioned before, the laser dye has a broad fluorescence spectrum which will be an important requirement for the generation of short pulses. A 40 fs pulse requires a frequency bandwidth of 366 cm^{-1}. If it is to be centered at 620 nm the half intensity wavelengths should are 613 and 627 nm. Such a laser cannot be considered monochromatic source. The dye laser will lase in one or several modes depending on the alignment and pumping power. A method is needed to 'lock' the different modes into a coherent pulse. This locking can be achieved by active or passive methods. Only passive mode-locking has successfully generated stable pulses with < 40 fs duration.

For a given laser cavity, in this case a ring, the time the light takes to make a round trip τ is simply determined by:[11]

$$\tau = L/c \qquad (1)$$

where L is the round-trip length of the resonator and c is the speed of light (the index of refraction is taken to be unity for simplicity). A necessary condition for lasing is that the phase of the optical radiation is reproduced identically after every round trip within an integer number of 2π insuring the light does not interfere destructively with itself. This condition defines each resonance frequency of the cavity ν_n to be

$$\nu_n = n/\tau, \qquad (2)$$

where n is an integer. The actual frequencies of oscillation of a laser system will be those that are resonant with the cavity and which see a larger gain than intracavity loss. These frequencies are known as the longitudinal modes of the laser. A laser system can operate in a single frequency mode, in multimode and, as we will see, the laser can be mode-locked.

The concept of mode-locking can be easily understood for the case of only two modes. The intensity of the radiation field will equal the square of the sum of the electric field of each of the modes. From the square of the sum, we present only the cross term which carries the time dependent information and we simplify it in the following expression:[12]

$$I \approx \cos(\omega_1 t + \phi_1(t)) \cos(\omega_2 t + \phi_2(t)), \qquad (3)$$

where $\omega = 2\pi\nu$, and $\phi(t)$ represents the time dependent phase of each pulse. For multimode operation there is a random relation between the phases of the pulses. If we set the phases ϕ_1 and ϕ_2 equal to zero, we notice that equation (3) can be simplified considerably, and thus the intensity takes the form:[12]

$$I \approx \cos(\omega_1 t + \omega_2 t) + \cos(\omega_1 t - \omega_2 t). \qquad (4)$$

The first term oscillates at a frequency $2\pi(n_1+n_2)/\tau$. For our CPM system, the round trip length is 3.6 meters and the fundamental frequency is 83 MHz; this implies that frequencies ν_n near 620 nm correspond to $n \sim 10^9$. This makes the first term of Equation 4 oscillate too fast for detection. The second term, however, oscillates at a frequency $2\pi/\tau$. Since all resonant modes have frequencies related by an integer to the fundamental frequency of the laser, adding more modes maintains a maximum intensity every time τ. For two modes, the modulation is sinusoidal, but as the number of modes increases, the maxima begin to look like pulses. The formation of these pulses is all based on the condition that their phases are all equal to zero; this condition is otherwise called phase-locking or mode-locking.

The duration of the mode-locked pulse depends on the number of modes. For a Gaussian pulse of duration τ_p, the frequency width must be $\Delta\nu$ and the relation between τ_p and $\Delta\nu$ is given by:[12]

$$\tau_p \, \Delta\nu = 0.4413, \qquad (5)$$

and $\Delta\nu$ is linearly dependent on the number of modes, since they are equally spaced. Therefore, the more modes one can mode-lock, the shorter the resulting pulses. In the CPM cavity, the phases are locked by the use of a saturable absorber that functions as a very fast shutter which only opens once every round trip time of a single pulse.

The ring dye laser, before mode-locking, will produce a continuous laser beam. The pulses are generated by the introduction of the saturable absorber jet (SAJ in Figurel). The SAJ consists of a vertically flowing solution of 3,3'-diethyloxadicarbocyanine iodide2,3 (DODCI) in ethylene glycol. Its absorption spectrum is displayed in Figure 4. Both the gain and the saturable absorber jets are to be considered optical elements, thus must be optically flat and parallel. The gain jet is $\sim$ 100 μ thick and the saturable absorber jet (a commercial stainless steel nozzle compressed to size) is $\sim$ 40μ thick.[13] The flow through the nozzle must be perfectly steady with no vibrations or bubbles. These requirements led to the design and construction of "dye modules," consisting of a liquid pump, a filter for small particulates, a vibration dampener, and a dual purpose container which served as a temperature control and flow homogenizer. To insure inertness to the dye, the materials making up the dye modules are either 316 stainless steel (non-magnetic) or teflon tubing. Seven such dye modules were constructed in this laboratory with very successful performance.

The dye laser, when optimized for power, will preferentially run at about 590-600 nm with a relatively narrow bandwidth of ~ .2 nm. The second jet of ethylene glycol, and no added saturable absorber dye, may be introduced without disturbing the dye laser since the jet is transparent, flat, and the two surfaces are parallel. Like the gain jet, the absorber jet is introduced at Brewster's angle (calculated using $\theta_B = \tan^{-1}(n_2/n_1)$ for $n_{1,2}$ the refractive indices of air and ethylene glycol respectively, $\theta_B = 55^o$) in order to minimize the losses due to reflection from the surfaces of the jet. Once the jet is in place (at the focal point between mirrors A1 and A2, R = 5 cm see Figure 2) a concentrated solution of DODCI is slowly added, up to the optimum concentration. As the DODCI concentration rises, the laser wavelength shifts from the initial ~590 nm to 625 nm. The bandwidth of the laser changes from 0.2 nm to 10 nm. At this point, the CPM is mode-locked and produces 50 fs pulses at 80 MHz. One can follow these changes by sending one of the output beams of the laser onto a diffraction grating and displaying the spectrum on the wall.

The physics of pulse formation in the ring cavity consists of a very fast switching action that lets through only short bursts of light. This "shutter" is accomplished by the saturable absorber in the following way: Initially, when the available light in the cavity is very low, no light goes through the saturable absorber. This is true except when there is a fluctuation in the intensity due to noise, and a spike is formed. A spike can partially go through the absorber and is immediately amplified at the gain jet. An amplified spike will easily saturate the absorber every round trip and continue to amplify until the gain is saturated. At this point pulses are traveling inside the laser cavity. Increasing the absorber concentration favors the unique event when two counter-propagating pulses "collide" in the very thin saturable absorber jet. The collision achieves a higher intensity and therefore minimal losses at the saturable absorber. Because of gain recovery restrictions, the optimum condition for the CPM laser exists only when two counter-propagating pulses are in the cavity. The position of the saturable absorber jet is not critical because of the ring design. Experiments on linear cavities require positioning of the jet within a few microns. The linear and non-linear pulse shaping mechanisms in the cavity which further shorten the pulses are considered in the following sections.

2.3 The prisms and group velocity dispersion

In theory, the pulses should shrink until the gain bandwidth becomes the limiting factor, for R6G this limit is about 8 fs. However, a CPM operating without prisms (P_1 - P_4 in Figure 2) will produce pulses that are about one picosecond in duration. The reason why the pulses are so broad in time is because there are materials in the path of the light that have indices of refraction that vary with frequency; therefore different frequencies traverse the cavity at different speeds.

A set of four prisms (P_1 to P_4 in Figure 2) is introduced in the laser cavity to compensate for the group velocity dispersion (GVD). GVD is explained by the fact that different wavelengths of light travel at different speeds in a given media. In other words, the refractive index (n) of an optical element varies as a function of wavelength. This variation of $n(\lambda)$ is of particular importance for ultrafast pulses where the bandwidth of the pulses is ~10 to 20 nm. The performance of a CPM laser without correction for GVD will be limited by the total GVD caused by its optical components (mainly the mirrors and the jets). Initially CPM lasers had a reputation of not being reproducible. The same laser behaved dramatically different by exchanging one of the internal mirrors with another, even when the mirror had the same reflective characteristics. The lack of reproducibility was due to very small differences in the dielectric coatings of the mirrors, which caused large changes in the internal GVD of the resonator.

The variation of the refractive index as a function of wavelength is a manifestation of the fact that light travels slower in any media other than in vacuum. The "slowing down" of the light, as it travels through a transparent medium such as glass, is caused by the interaction of the atoms with the electric field. The closer the wavelength is to being resonant to a transition in the medium, the larger the derivative of the refractive index with respect to the wavelength. In glass, this occurs at wavelengths shorter than 200 nm. An ultrafast laser pulse can be thought of as a packet of photons or wave packet having a spread in wavelength ($\Delta\lambda$) and a group velocity (v). When such a pulse is allowed to travel through a medium whose refractive index changes with wavelength, the pulse experiences a broadening effect caused by the delay of its short wavelength components with respect to the longer wavelength components. This broadening effect is called group velocity dispersion.[5]

In theory, a pulse affected by a given amount of positive GVD can recover its original temporal shape by traversing a material exhibiting a variation of its refractive index with wavelength which is opposite to that of the material that caused the GVD, namely, negative GVD. A material will cause negative GVD when the incident light is close to resonance with a transition. Looking for a material that can compensate for GVD is not very practical, since there are very simple optical arrangements that will have a total negative dispersion. One of the most common arrangements is the four prisms in the CPM cavity as shown in Figure 2. The idea is very simple: in normal dispersion, bluer wavelengths are delayed with respect to redder wavelengths. To compensate we create a negative dispersion arrangement (more about this in Section 4) in which the beam path for bluer wavelengths is shorter, thus letting those components "catch-up" with the redder components of the pulse.

The GVD of the special mirrors,[15] and the GVD of the very thin gain and saturable absorber jets[16] (100 and 40 μ respectively) is corrected by the four intracavity prisms.[9] Translation of any of the prisms along the bisecting angle serves to regulate the total amount of dispersion. Achieving the optimum dispersion is done by displaying the spectrum of the CPM output on a wall. The optimum intracavity dispersion is reached when the spectrum is broadest, corresponding to the shortest pulses. More details about correction of GVD and other effects will be given in Section 4.3.

2.4 Operation the CPM laser

The CPM as described will perform surprisingly well, compared to other laser systems that generate ultrashort pulses (for example, the femtosecond synchronously pumped dye laser systems described by Johnson and Simpson[17]). The CPM will run for about two months before the saturable absorber dye DODCI degrades. Changing the DODCI (a one hour task) will be enough to get the CPM to run again. The shortest pulses ever recorded out of a CPM are ~ 27 fs, and recent developments have given it some wavelength tunability.[9] Alignment of the CPM is usually not needed except for some minor adjustments during the dye changes. Part of the success of our laboratory has been due to the stability of the resonator.

The output of the CPM consists of two beams. One beam is generated by a pulse travelling clockwise in the cavity and the other by a pulse travelling counterclockwise. The clockwise pulse train is the one used for the experiments, since the counterclockwise pulses traverse the gain jet before exciting the cavity, and are therefore broader. The beam which is not used for the experiment is dispersed by a grating and displayed on a wall as a visual monitoring aid that allows one to tweak the CPM.

The description of the CPM given here is simple, with no attempt made to present the theory of pulse shaping occurring in the cavity. For a more complete treatment of this subject, the reader is referred to the theoretical work of Martinez et al.,[18] Diels et al.;[19] and the experimental observations of Salin et al.[20] and Valdmanis et al.[9] Only a simplified version of the pulse shaping mechanisms will be given in order to introduce some of the effects that lead to ultrashort pulses.

The shortest pulses that can be produced in a CPM are, to first order, limited by the gain bandwidth and by the spectral characteristics of the saturable absorber. In addition, there are more pulse shaping effects active in the generation of ultrafast pulses in the CPM. It has been found that there is a source of negative GVD in the cavity besides the prism arrangement.[9] The extra source of GVD is due to the nonlinear effect of self-phase-modulation (SPM); its effect on a pulse is to lower the frequencies of the leading half of the pulse and rise those of the trailing half. The saturable absorber has been singled out as the main source of SPM. Experimental evidence shows that translating the absorber jet past the focal point results in positive dispersion. As the jet is translated, an optimal distance is found at the point at which further translation has the effect of negative dispersion. Valdmanis and Fork showed that the shortest pulses have been found to occur close to the turning point but towards the negative dispersion regime.[9] A behavior similar to the one described is qualitatively consistent with our CPM.

The prism arrangement serves to balance the positive GVD caused by the mirrors and the negative dispersion caused by SPM in the saturable absorber jet. SPM has been used as a pulse compressing technique and it is believed that a similar pulse compression effect is taking place in the CPM[9] when it is operating in the optimal regime (more about pulse compression is included in Section 4). Since the effect of SPM on a pulse is lowering the frequencies of the leading half of the pulse and raising those of the trailing half, the emerging pulse is spectrally broader. Linear GVD compensation in the prism arrangement will allow the trailing part of the pulse to "catch up" with the leading part. In this way, the pulse tends to collapse upon itself. The mathematical form of the equations describing a CPM arrangement that balances the effects of SPM, GVD, saturable absorption and saturable gain has led to consideration of a "soliton-type shaping mechanism" yielding a squared-hyperbolic-secant (sech^2) pulse shape.[18,19] Evidence for the soliton character of the pulses from a CPM laser is increasing. Salin et al observed high order solitons directly produced by a CPM.[20] For a simple, yet insightful review of solitons, the reader is referred to the paper by Mollenauer and Stolen.[21] The fundamental soliton is the mathematical solution that has zero change in its shape after propagation through the linear and nonlinear pulse shaping mechanisms in a laser cavity. Physically, it corresponds to a pulse of the right amplitude (shape) that is unaffected by the net effect of pulse shaping caused by positive and negative GVD in the prisms and the nonlinear SPM in the absorber jet. The output pulses of CPM arrangements are best fitted to sech^2 pulses, which are considered to be additional evidence of the soliton-like behavior. To summarize the pulse shaping effects of the CPM, Figure 5 shows a schematic drawing of the pulses as they traverse the different components of the CPM.

3. Amplification of femtosecond pulses

The average output of the CPM laser is 20 mW, with a repetition rate of 83 MHz and peak energies of 240×10^{-12} joules per pulse. Unfortunately, this amount of energy per pulse is insufficient to do the experiments we are interested in. The losses involved in preparing the pulses at the wavelengths of interest result in attenuations of about a factor of 100, leaving about 2 pico joules for the experiment. Amplification of the femtosecond pulses by a factor of 10^6 is therefore necessary. Such amplification is possible and was performed with the setup described below.

3.1 The concept of amplification

The concept of amplification of laser light is simple. A laser beam travelling through a medium that has undergone population inversion will extract a portion of the available energy according to the simplified formula:

$$I(z) = I_0 \exp\{\gamma(\nu)z\}, \qquad (6)$$

where I_0 is the initial intensity, $\gamma(\nu)$ is the gain of the medium as a function of frequency and z is the path length. At each amplification stage, the pump light is absorbed by the dye and a situation of population inversion occurs. The CPM output, when travelling through these gain stages, extracts the energy and is amplified. This is the simplistic view of the process. In reality, there are several complications in a femtosecond laser amplifier: saturation, amplification of spontaneous emission (ASE), and amplification of the leading edge (ALE) are discussed in more detail.

3.2 Saturation effects

Equation 6, is basically a form of Beer's law if we think that each encounter of a photon with a molecule leads to gain as opposed to absorption. The gain arises because we assume that the media is in a state of population inversion. As expected, the gain a laser pulse can experience is not infinite because of saturation. For simplicity, consider the initial intensity I_0 to be related to the number of molecules in the amplification path of the laser. When the initial pulse energy I_0 is very small, say $I_0 << 1$, then Equation 6 holds, since most molecules are unlikely to see a single photon. When $I_0 >> 1$, then most molecules see many photons, the first, however, causing saturation. How the gain I/I_0 is related to the path length z and the initial energy I_0 is shown schematically in Figure 6 taken from reference 22. We can see that it is important that the input beam have a relatively low intensity in order to avoid saturation. These effects are taken into account in the design of a laser amplifier. The solution is not simply to avoid saturation completely, since one wants to minimize the loss at each stage. The design of the amplifier must take into account saturation and loss plus other problems which are more dynamic in nature as will be discused below.

An amplifying medium can operate in three distinct regimes: (i) unsaturated, (ii) lossless saturated regime, and (iii) loss-limited saturated regime.[23]

3.2.1 The unsaturated regime

A pulse of light traveling through an amplifier medium in the unsaturated regime experiences a gain $\gamma(\lambda)$ which is dependent on wavelength λ. The gain curve of the dyes in the amplifier is not flat in the spectral region of the CPM pulses, therefore the pulses experience a wavelength dependent gain. Further complicating the pulse distortion is the fact that the pulses are also affected by a wavelength dependent absorption $\alpha(\lambda)$. From the analysis of the fluorescence spectra of the dyes used in the amplifier (shown in Figure 8 from a study by Drake et al.[24]), the best amplification range is from 620 to 640 nm. Empirically we have found that $\alpha(\lambda)$ increases sharply from 613 nm to shorter wavelengths. Propagation of pulses through an unsaturated amplifying medium results in the narrowing of the pulse's spectrum. This in turn implies that the time duration of the pulse, related to the spectrum by transform theory, will be broadened. The first and second stages of the amplifier are run in this regime in order to avoid other complications described below. Thus, the dye in the first stage must have very small absorption of the CPM laser (see in Figure 8 the absorption cross section, as a function of wavelength, of both dyes). This is why Kiton Red is used in the first stage.[25]

3.2.2 The lossless saturated regime

This is the ideal regime for the last two stages of the amplifier. This regime corresponds to a homogeneously broadened laser medium. The gain in the central portion of the peak of the pulse saturates, therefore the sides of the pulse are amplified preferentially. The resulting amplifying medium has a smaller amplitude and a broader gain curve. This cancels the narrowing effect of the unsaturated case. In this manner, wavelength dependent gain balances with the wavelength dependent loss resulting in a balanced overall gain profile.

3.2.3 The loss-limited saturated regime

When the saturation of the medium is severe, a process called hole-burning becomes predominant. Hole-burning is the process of complete depletion of the gain at the wavelength of saturation. This leads to inhomogeneous broadening of the gain medium and thus of the pulses. This condition is to be avoided in an amplifier, because inhomogeneous broadening cannot be corrected for.

3.3 Amplification of spontaneous emission

The first complication inherent to the process of amplification is the amplification of spontaneous emission (ASE). By spontaneous emission, we mean the normal fluorescence caused by the excitation of the laser dye. This fluorescence, if in the direction of the rest of the stages, will see a tremendous amplification. As the spontaneous emission is further amplified, it depletes the amplification stages thus reducing the gain that the CPM could receive. If this situation is unattended, the resulting pulses will resemble the spectrum of the spontaneous emission of the dyes used and the pulse duration will be that of the pump source (in our case 2 ns). The reason why spontaneous emission is preferentially amplified is because it persists for 10^5 times longer than the femtosecond pulses we are amplifying.

The first rule of amplifier design is to avoid surfaces perpendicular to the propagation of the beam in order to avoid additional laser cavities. To avoid ASE, one includes spatial filters, restricting the passage of light not having the exact spatial characteristics as the light of interest. The spatial filter consists of a pair of positive lenses (thus having a focal point between them) and a diamond pinhole with a diameter of 2 thousands of an inch. The spatial filter is initially aligned with the CPM going through. The size of the pinhole is calculated to be large enough for the CPM's waist to go through ~95%.

In our amplifier (see Figure 7) pinholes P1 and P_2 are placed after the first and third stages respectively to spatially filter ASE. In addition, a saturable absorber jet (SAJ) after the second stage is used to temporally filter the ASE. Since the femtosecond pulses are a factor of 10^5 shorter in time, their peak power is proportionally higher. The difference in peak power allows one to make a saturable absorber of malachite green in ethylene glycol which only lets amplified femtosecond pulses through and blocks the ASE.

3.4 Amplification of the leading edge and non-linear effects

Amplification of the leading edge of the pulse (ALE) is a complicated process which is very hard to avoid. The leading edge of a pulse travelling through a lossless saturated stage will see less saturation compared to the rest of the pulse. This will cause a distortion of the pulse's amplitude or, in the worst and more realistic case, the pulse, having experienced some GVD, will have preferential amplification of the wavelength that is leading (redder for positive dispersion and bluer for negative). To avoid ALE, one basically has to avoid saturation, but there is always some degree of saturation in the amplifier. In order to minimize the effects of ALE, saturable absorber jets are used.[26] A saturable absorber will preferentially absorb the leading edge of the pulse. If the absorption cancels ALE then the problem is cured. Matching the wavelength dependent absorption of the absorber jet to the wavelength dependent gain caused by ALE is non trivial and what works for positively chirped pulses (pulses affected by positive GVD) will not work for pulses negatively chirped.

From the previous discussion, it would seem that the ideal location for the saturable absorber would be after the third or fourth stages. In practice, the intensity of the pulses after the third stage is already too high and the pulses get very distorted when focused in the absorber due to non linear effects as discussed below. The saturable absorber in our amplifier is thus placed after the second stage, and in a sense, it trims the leading edge of the pulse just enough to precompensate for subsequent ALE in the last two stages.

Other mechanisms that distort the pulse shape during high gain amplification are those that depend on higher order terms of the refractive index causing further irreparable damage. In the last stages of the amplifier, the pulse energy is of the order of 5 μj and the pulse duration is of the order of 10^{-13} seconds, giving a peak energy of 5 x 10^7 watts (fifty megawatts!). The spot sizes are of the order of 2 mm in diameter in the gain cells so the peak energy density is ~2 x 10^9 watts/cm2. At these energy densities, nonlinear effects such as nonlinear-frequency-generation, self-phase-modulation and self-focusing occur. Since the spectral bandwidth of the laser is related by a Fourier transform to the temporal width of the pulses, any distortion of the pulse's spectrum will transform as pulse broadening. The amplifier, as it is currently operating, has successfully balanced all of the parameters in the equation and is capable of delivering a 10^6 overall gain with a good mode, smooth spectrum, and good pulse-to-pulse stability.

3.5 Description of the amplifier

The amplifier (see Figure 8) was built following guidelines of the design by Fork et al.[27] It consists of two sets of optics: those that guide the output of the CPM, and those that guide the pumping laser (a DCR3-G Yitrium-Aluminum-Garnet "YAG" solid-state laser). The CPM optics are; the four stages of amplification Cl to C4, and prior to each stage there is a pair of lenses acting as telescopes that collimate and adjust the beam radius. Between the lens pairs there may be a saturable absorber jet SAJ or a diamond pinhole (PH). The YAG optics include one lens pair that functions as a telescope. In this case the beam does not go through a focus between the lens. This is to avoid air breakdown at the focal point due to the high energy densities. Air breakdown occurs when the energy density is enough to ionize the nitrogen in air mainly by multiphoton ionization.[28]

There are three beamsplitters. The first beamsplitter divides the beam and reflects 70% of the light to the fourth amplification stage, the remaining light is then expanded in order to fill the path length of the first three stages which are side pumped. Of the total energy, the first three stages get: 3%, 3% and 24% of the YAG beam respectively. Minor adjustments are usually made by introducing neutral density filters. Three cylindrical lenses CL1 to CL3 focus the YAG onto stages 1, 2 and 3. The focussing of a cylindrical lens is only in one plane therefore, at the focal point, a narrow slit is formed. The resulting slit length equals the beam diameter before the cylindrical lens and it is adjusted by the telescope to equal the path length of the cell. The slit width is matched to the oscillator's beam diameter at each stage.

The CPM pulse train is steered from the CPM to the amplifier. The beam goes through the first telescope where its collimated to a diameter of 0.5 mm. After the first stage (Cl) the beam encounters subsequent telescopes and amplifying stages (C2 - C4). The beam diameter is increased from 0.5 mm in the first stage by a factor of two thereafter in each telescope. The increase keeps the peak power from achieving saturation and nonlinear effects.

The dyes in the amplifier are the same as those used by Fork et al.[27] Kiton Red in the first stage and Sulforhodamine 640 in the other three stages. Water is the solvent in all amplification stages because the refractive index of water depends less on temperature than that of methanol or ethanol. The use of water solutions of dyes tends to reduce instability caused by nonuniform heating of the active region. Since rhodamine dyes have been found to dimerize in aqueous solutions Amonix LO (Lauryldimethylamine oxide) is used as a surfactant to deaggregate the dimers and to increase the efficiency of the dye by quenching the triplet state of the dye molecules.[29] The performance of the amplifier as described in this section is discussed below.

3.6 Operation of the amplifier

The CPM repetition rate is 83 MHz and the maximum repetition rate of the YAG laser is 20 Hz, so unfortunately, only one out of every four million pulses is amplified. A synchronizer module (Spectra Physics SM1) was used to delay the trigger of the YAG laser until both a pulse from the CPM and a trigger from the YAG laser arrived at the module. A variable delay allowed sweeping the time difference between arrival of the two pulses, and was used to overlap the two pulses at the gain stages. The CPM pulses used for synchronization are extracted from the ~1% of the clockwise transmission of the CPM B mirror (see Figure 2). The transmitted laser light is focused into a fast photodiode which converts light pulses into electric pulses. These pulses are sent to the synchronizer module. The placement of the amplifier optics maintains the synchronization of the two lasers throughout the four stages of amplification.

The first two stages of the amplifier have to be operated in the unsaturated regime, therefore the pulse spectrum is narrowed. By running the third and fourth stages in the saturated lossless regime, some of the lost bandwidth is gained back. The amplifier run at the typical pumping energy of 300 mJ produces pulses with ~500 μJ of energy and time duration of ~600 fs. The temporal broadening is caused by the femtosecond pulses travelling through ~10 cm of water and ~ 4 cm of quartz resulting in a large GVD (as explained in the following Section 4). A prism pair is used before the amplifier to precompensate the GVD, and a second prism pair arrangement is used after the amplifier in order to introduce negative GVD to the pulses in a more symmetrical fashion. The chirp compensated pulses are then of ~ 50 fs in duration.

4. Pulse compression

The principle of pulse compression originated during World War II with chirp radars, when microwave transmitters would broadcast long chirped pulses and the receivers compressed them to achieve a much sharper range resolution.[11,30] Compression of optical pulses using concepts based on microwave technology were proposed in 1964 by Gires and Tournois[31] and later by Giordmaine et al.[32] By 1969, Treacy proposed a simple method for optical pulse compression with diffraction gratings[33] and Fischer et al. found that propagation through a Kerr medium could impress a linear chirp on an optical pulse.[34] By 1970, all the elements for optical pulse compression were available, yet, it wasn't until 1980 when Mollinauer et al. observed picosecond pulse narrowing in optical fibers.[35] Shank et al.[6] used a optical fibers and a grating pair to compress a 90 fs pulse down to 30 fs, and eventually using various pulse compression techniques[36] achieved 6 fs.[7]

For our experimental setup, we are interested in generating 40-50 fs pulses and thus do not need optical fibers. In order to describe the pulse compression techniques that we utilize in our laboratory, it is important to first review the propagation of a laser pulse through transparent media and to define group and phase velocity. The concept of dispersion is then introduced and is followed by the concept anomalous, or negative, dispersion and optical instruments with variable anomalous dispersion. Finally, we show how anomalous dispersion is used compensate for dispersion and thus achieve pulse narrowing.

4.1 Group velocity dispersion in transparent media

A short optical pulse can be thought of as a collection of waves with different frequencies ω_i. As a light pulse propagates through a transparent medium, the velocity of each component wave (called the phase velocity) is given by v_ϕ. In vacuum, $v_\phi = c$, but in optical media, $v_\phi = c/n(\omega)$, where $n(\omega)$ is the index of refraction characteristic of that material as a function of frequency. Traversing a block of quartz, for example, makes each wave propagate at a different phase velocity. This in turn causes the overall envelope to move at a velocity v_g (group velocity) different than the central frequency phase velocity in all media except vacuum.

The effect of a linear optic on a pulse of light is usually determined by the phase shift ϕ, in radians, caused by the propagation.[11,34] The phase shift is defined as $\phi(\omega) = -(L/c)\omega n(\omega)$, where L is the length of the media. By intuition, one can check that propagation through an $L = \lambda$ in vacuum causes a phase shift of $\phi(\omega) = -(L/c)\omega$. Given that $\omega = 2\pi c/\lambda$, we confirm that $\phi(\omega) = -2\pi$, as expected.

The phase shift can be best understood when expanded around the central frequency of the pulse, ω_o, such that:

$$\phi(\omega) = \phi(\omega_o) + (\omega-\omega_o)\phi' + (\omega-\omega_o)^2\phi''/2 + \cdots, \qquad (7)$$

where $\phi' = (\partial\phi/\partial\omega)_L$ and $\phi'' = (\partial^2\phi/\partial\omega^2)_L$.
The first term of Equation (7) is a constant determined by the central frequency of the pulse ω_o. The phase velocity is then $v_\phi = -\omega L/\phi(\omega_o)$. The second term determines the group velocity given by $v_g = \omega/\phi'(\omega)$. As long as $\phi'(\omega)$ remains a constant, the group conserves its shape and is not broadened; it is only delayed. By taking the derivative we note that $\phi' = -(L/c)[n(\omega)+n'(\omega)]$ and therefore the larger the index of refraction, the slower the group velocity.

When ϕ' is not a constant (i.e. $\phi''(\omega) \neq 0$, or equivalently $n''(\omega) \neq 0$), the group velocity varies as a function of frequency within the pulse causing a frequency sweep (also called a chirp). As the pulse is chirped, the envelope is broadened. The group dispersion is the third term of Equation (7) and can be found by direct derivatives, obtaining as a function of frequency:[37]

$$\phi'' = -\frac{L}{c}\{2n' + \omega n''\} \tag{8}$$

or as a function of wavelength:

$$\phi'' = \frac{L\lambda^3}{2\pi c^2}\, n''. \tag{9}$$

Higher order terms of Equation (7) also cause pulse distortion; in particular, the cubic term was found to be the determining step in compressing pulses down to the record 6 fs.[7] Since we are not interested in pulses shorter than 40 - 50 fs, we can neglect these higher order terms.

Equation (9) is useful in calculating the pulse distortion caused by propagation of femtosecond pulses through transparent media. For Gaussian pulses, one can calculate that a pulse with FWHM τ_{in} is broadened by the following ratio:[37,16]

$$\frac{\tau_{out}}{\tau_{in}} = \sqrt{1 + f^2\left(\frac{\phi''}{\tau_{in}^2}\right)^2}. \tag{10}$$

where $f = 4\ln 2$. This result shows that GVD is not caused directly by a large index of refraction, but by a large n''. The values of n, n' and n'' of some common optical media are given in Table 1 at 620 nm (for a reference containing these parameters for more than thirty compounds see Bor and Rácz[39]).[38]

To illustrate the broadening of femtosecond pulses, we take our amplifier as a first example. Along the beam path, a femtosecond pulse from the oscillator traverses through 4 cm of quartz lenses and plates and 10 cm of dye solutions which we approximate as pure water. Taking the numbers from Table 1, we calculate $\phi'' = 7630$ fs^2. The effect of this large GVD factor on femtosecond pulses is presented in Figure 9 which is a plot of the resulting pulsewidth, τ_{out}, as a function of the input τ_{in}. As can be seen, for pulses shorter than 100 fs, there is a substantial broadening, while for pulses longer than 150 fs the broadening is quite small. These considerations are very important in the design of these femtosecond amplifiers; note that a 50 fs pulse is broadened almost by a factor of eight, while a 200 fs pulse experiences a 10% broadening. The dependence of GVD on the pulse duration is what makes the generation of sub-picosecond laser systems more complicated. For example, a 1 mm quartz plate doubles the pulsewidth of a 6 fs pulse. In order to compensate for the GVD one must find an optical medium with a negative (anomalous dispersion) ϕ'' (transparent materials have usually positive GVD).

4.2 Anomalous dispersion

Anomalous dispersion is a well known phenomena in optics. Usually it is associated with the propagation of light near absorption resonances in the optical media. The application of anomalous or negative GVD for the generation of very short pulses required finding a transition that occurred in the precise region of the spectrum that corrected for the positive GVD that had caused a given pulse to broaden. In early experiments, Shank[30] tried to propagate the pulses through atomic vapors having the necessary transitions to cause negative dispersion before using the grating pair as proposed 10 years earlier by Treacy.[33] The reason for the success of grating pairs is that they provide the experimenter with a variable ϕ''. More recently, prism pairs have become a more popular pulse compression technique as will be discussed later.

4.2.1 Grating pairs

The grating pair arrangement as first proposed by Treacy[33] is shown on Figure 10. The reason why it has a negative GVD can be simply explained by the fact that longer wavelengths travel a longer path than the bluer wavelength and are thus delayed. This is, in fact, the opposite effect occurring in most transparent optical media. The expression for ϕ'' for the arrangement shown in Figure 10, is the following:

$$\phi_G'' = \quad - \frac{L\lambda^3}{2\pi c^2} \frac{m^2}{a^2} \frac{\mathcal{L}}{\cos^3\theta} \tag{11}$$

where a is the gratings groove spacing, m the diffraction order used, $\mathcal{L}$ the distance between the gratings, and θ the angle between the normal to the grating and the diffracted beam.

The reason why gratings are not the most common pulse compression tool is that they represent a very large loss, usually larger than 30 %. Furthermore, gratings can only provide negative GVD and in some cases one may want some positive GVD. Only an arrangement of equilateral prisms can provide high throughput > 90% and variable positive and negative GVD as discussed below.[40]

4.2.2 Prism pairs

An arrangement of four equilateral prisms can be achieved in which the path of the longer wavelengths is redder than that of the bluer wavelengths and thus negative GVD is achieved. The advantage over the gratings is that Brewster's angle at 620 nm is near 60° for glass, therefore the losses are minimal. An additional advantage is that one can regulate how much glass the beam traverses and can therefore balance positive and negative GVD.

The prism pair arrangement is shown in Figure 11. Note that the arrangement is very practical because the prisms at the extremities can be translated outside the beam path without need for realignment of the original optical setup. Furthermore, one can regulate the amount of glass that the laser goes through by translating any of the prisms along the bisecting angle, as shown on the insert.

An expression for ϕ'' of the prism arrangement shown was originally calculated by Fork et al.[40] and is now simplified to:[38]

$$\phi_p'' = \frac{4\lambda^3}{\pi c^2}\left\{-Ln'^2 + e\left[\frac{nn''}{1+n^2} + n'^2\left(1 - \frac{1}{n^2(1+n^2)}\right)\right]\right\} \qquad (12)$$

where n, n', n" and λ are all expressed at the central wavelength of the pulse λ_0, L is the distance in millimeters BC between the prisms and the quantity e is the amount of glass the beam traverses and is defined as e = AB + BC, in millimeters (see Figure 11 insert).

For practical purposes, one can simplify Equation (12) and make it specific for a given wavelength and optical material. In our case, the dispersion of our amplifier is quite large and a highly dispersive optical material is needed. We have chosen SF10 (Schott[41] flint glass #10) prism in a double pair arrangement as shown in Figure 11. The central wavelength of the pulses after amplification is 620 nm. With this information we obtain for ϕ_p'':

$$\phi_p'' = 2(836.1e - 39.47L) \qquad (13)$$

where e and L have been previously defined and ϕ_p'' is in fs^2. Note that the positive dispersion caused by the glass is regulated through the distance e and the negative dispersion caused by the optical arrangement is controlled by the distance L. The factor of 2 is included because of the two prism pairs.

With equation (13), the GVD caused by the amplifier and other optical elements in the beam path can be compensated for. For our amplifier we can calculate the distances for our prism arrangement.

In practice we have done this, but it turns out that there are other effects which must be taken into account as discussed below.

4.3 Compression by GVD compensation

Using Equation (13) we calculate we need a prism spacing L = 52.0 cm for a parameter e = 2.0 cm, in order to correct for ϕ'' = 7628 fs^2 of our amplifier. Experimentally, the spacing we find to optimize the generation of very short pulses is 42.5 cm, however, this number is very sensitive to the parameter e since every change of a millimeter in e implies a change of 21 mm in L. Another reason for the discrepancy in this calculation is that we assumed that the amplifier cells were filled with water instead of dye solutions. The absorption of the dye solutions causes changes in the indices of refraction, which in turn change the GVD. Furthermore, the high intensities of the pulses cause saturation and non-linear processes such as self phase modulation which distort the pulses in a more complex way.

In our laser system we have found that the output of the CPM consists of negatively chirped pulses which are compensated with an SF10 prism pair with L = 10 and e = 1.8 cm. At this point, the pulses are ~ 100 fs FWHM. In the amplifier the pulses suffer some dispersion and some SPM which gives an output of ~350 fs. Compression with the two prism pair arrangement described above with L = 42.5 and e = 2.0 cm yields pulses that are ~ 50 fs FWHM. The fact that the pulses are shorter after amplification implies that there is some SPM in the amplifier. It is expected that the SPM is caused in the fourth stage where the largest intensities are generated.

Operationally, optimization of all prism pairs in the laser system must be done as follows. (i) One must determine the FWHM of the desired pulses and make sure that the output spectrum of the amplified pulses has the necessary bandwidth (not more and not less). One must be careful that the output of the amplifier is not distorted by severe saturation or by ASE. The bandwidth is controlled by translating one of the intracavity prisms in the CPM. (ii) Once the bandwidth is fixed, an iterative process begins to determine the optimum position of the prisms before and after the amplifier. The prisms before the amplifier compensate the chirp of the CPM and can precompensate for GVD in the amplifier. The prisms after the amplifier correct for the GVD caused in the amplifier. By iteratively optimizing one set of prisms while slowly scanning the other set along a given direction, we have been able to generate transform limited pulses with widths ranging from 200 to 50 fs. There are some technical difficulties with this procedure, the most important being: making sure that prism translation does not displace the beam, and making sure the laser is stable enough to allow at least 2 hours without any change in the output of the CPM.

5. Pulse characterization

The direct characterization of laser pulses shorter than a few picoseconds is not currently possible because the state of the art electronic devices have longer response times than the times we are interested in. The only event as short as a femtosecond pulse is a similar femtosecond pulse, and therefore, these pulses are measured using correlation techniques where a pulse is compared to itself or to a similarly short pulse. These kinds of correlation techniques have been used since the early generation of picosecond pulses. In fact, Maier et al. used an intensity correlator based on second harmonic generation, SHG, in 1966 to measure 30 ps pulses.[42]

The characterization of femtosecond pulses requires a measurement of the pulse duration and an estimate of the coherence in the pulse. An intensity autocorrelation can give an estimate of the pulse duration, but will yield no information of the pulse shape or the coherence. Combined with a frequency spectrum of the pulse and assuming a pulse shape, one can estimate the extent of the coherence by calculating a pulsewidth-bandwidth product and comparing it to the theoretical value for a perfectly coherent pulse. A coherent pulse is said to be transform limited. New autocorrelation techniques have been proposed where one can obtain phase information of the pulses and thus directly measure chirp.[43] In this section, both intensity and interferometric autocorrelations will be discussed.

5.1 Intensity autocorrelation

The autocorrelation method relies on the use of a Michelson interferometer which separates and recombines the laser pulses at a specific point in time and space, see Figure 12. A second-harmonic generation (SHG) crystal is placed at the point at which the pulses are recombined. The SHG intensity will depend on the square of the intensity of the laser field in the crystal. The autocorrelator is set in a way that SHG will occur only when two pulses of light are overlapped. Scanning the interferometer delay on one of the arms will cause one of the pulses to move with respect to the pulse on the static arm. This way the overlap region of the two pulses is scanned and a map, or autocorrelation function (AC), of the intensity of the SHG is obtained.

In order to extract the pulsewidth from an autocorrelation function, one must convert the curve obtained, $I^2(D)$ vs. D, into a curve of I(t) vs. t, where $I^2(D)$ is the square of the intensity of the light as a function of D, the distance scanned by the pulse. The conversion of distance to time is trivial, but the conversion of the intensity is not. The AC function C(t), is represented by the integral

$$C(t) = \int_{-\infty}^{\infty} I(t)I(t-\tau) \ , \tag{14}$$

where τ is the delay between the pulses and I(t) is the actual pulse shape. The phase information of the pulses is usually lost in this kind of autocorrelation because it occurs in a much shorter time scale (in the next section we show how to preserve the phase information).

Hence, the pulse shape, which is determined by the phases, cannot be reconstructed from C(t).

As mentioned before, to characterize a femtosecond pulse we must obtain the spectral bandwidth and ascertain what part of the bandwidth is a coherent component of the pulse. The bandwidth is easily measured from a spectrum of the pulse, but in order to calculate the product between pulsewidth and bandwidth, $\Delta t \Delta \nu$, we must assume a functional shape for the pulses. Once we chose a functional form, we can then fit the autocorrelation and the spectrum and determine the product. Analytic expressions for C(t) have been obtained for a number of pulse shapes. They are reproduced in Appendix 1, taken from Reference 44. Femtosecond pulses have been found to usually fall between two categories, Gaussian and hyperbolic-secant-square (sech^2). The shape giving the best fit is used to measure the FWHM of the pulse. The formulas in Appendix 1 give the theoretical minimum for $\Delta \nu \Delta t$ for each functional form.

The autocorrelator we use is shown in Figure 12. Notice that this design insures that both arms traverse the same amount of glass. The beamsplitter (BS) is a very thin quartz substrate (200 μ) that has a 50% reflective coating, thus divides the beam into two arms. One of the arms is static and consists of a corner cube (CC). The other arm consists of a second corner cube mounted in a computer controlled micropositioner (P). The two pulses are recombined in the BS and focussed by a lens (L1) into the SHG crystal. The SHG light generated is collimated by (L2), separated from the fundamental by a UG11 filter (F), and detected by a photomultiplier tube. The arrangement can be collinear (non-zero background) or non-collinear (zero background).

In Figure 13 autocorrelation traces are shown for pulses directly from the CPM laser and for pulses after amplification and compression. In both cases, the pulses are fitted to hyperbolic secant squared functional forms and are found to be near transform limited.

5.2 Interferometric autocorrelation

The interferometric autocorrelation (IC) technique first proposed by Diels et al.[43] and Kurobori et al.[45] is very similar to the intensity autocorrelation and in fact both kinds of autocorrelations can be performed using the same optical arrangement. The principle is based on a collinear autocorrelation, but includes scanning with a much higher accuracy. Each feature on the IC is separated by ~2 fs and this is why for interferometric autocorrelations, it is very important to prevent any vibrations or air currents that can cause the phase information to be lost. It is also very important that the laser train is very stable.

In Figure 14, the IC of a 6 fs optical pulse from Fork et al.[7] is shown. Notice the very small number of features. This pulse is just 3 wave cycles in duration. These kinds of autocorrelations are very sensitive to chirp and to the coherence of the laser pulses. For some examples, see Reference 43.

6. Non-linear generation of different wavelengths

After compression of the amplified laser pulses, the beam is split and two arms can be defined. The first arm will become the pump laser which should be prepared at the wavelength which initiates the dynamic event of interest. The second arm defines the probe laser which is prepared at a wavelength that can spectroscopically monitor the evolution of the dynamics under interrogation. In the first beam-splitter, the two arms are generated with no delay between them and once combined in the experimental cell they will have a net zero-delay. The methods for the generation of the different wavelengths are shown in Figure 15 and are described below.

6.1 Second harmonic generation

Second harmonic generation (SHG) is a technique that is most commonly used in our laser system because of its simplicity and high efficiency. In essence, two identical photons, of frequency ω, combine to generate a photon of twice the energy 2ω. Since its first observation by Franken et al. it is the most common technique for the generation of uv pulses.[46] In particular it is used as part of the intensity autocorrelator. This technique is very simple, but special considerations for femtosecond pulses are used due to their broad bandwidth. If the SHG crystal is too thick, not all of the bandwidth of the pulse is transformed and the crystal functions as a filter. If the pulse looses bandwidth it gets proportionally broader in the time domain. Another difficulty is that at very high intensities, other non-linear effects occur in the crystal and the pulse gets broader in frequency and time.

In our laboratory, see Figure 15a, we have successfully used KDP and KD*P crystals for type I phase matching[47] with a condition $\Delta k = |k_{SH} - 2k_F|$, where Δk must be near zero for efficient conversion and k_{SH} and k_F are the propagation vectors of the second harmonic and the fundamental, respectively. The efficiency of SHG for even the thinnest of our crystals (100 μ) is better than 15%, this is to be expected since femtosecond pulses have very high peak intensities. The reasons why very thin crystals must be used depend on the short pulse duration and the broad bandwidth.

Since the pulses are very short, and there is a substantial difference in the group velocity between the fundamental and the second harmonic pulses, the interaction length in the crystal is not very long. To insure that the pulses are not broadened one must chose a crystal thickness such that the group delay is smaller than the pulse duration. The effect of pulse duration versus crystal thickness has been the subject of several studies recently reviewed by Laubreau.[48]

Comly and Garmire, in their analysis, define a unitless quantity M which can be interpreted as $M \approx L/L_{max}$, where L is the crystal thickness and L_{max} is the maximum crystal thickness which causes no pulse distortion.[49] They also define a unitless time T which is essentially $T = \tau_{out}/\tau_{in}$. The broadening of the pulses as a function of crystal thickness is shown in Figure 16 from Reference 49. As can be seen, the pulse broadens for $M > 1$, and suffers a significant distortion due to saturation for $M > 2$. Clearly, if one wants to avoid distortion of the pulse it is important to choose very thin crystals. Unfortunately, the conversion efficiency has an L^2 dependence, so it is important to compare the relative pulse amplitude and the pulse width as a function of the crystal thickness. In Figure 17 from Reference 49, these parameters are plotted. The pulsewidth starts at $T = \tau_{in}/\sqrt{2}$ and broadens as a function of M. Note that $M = 1$ is probably the best compromise where broadening is at a minimum and the relative pulse amplitude is 65% of the maximum. Notice also that by $M = 2$, the crystal essentially saturates and no further gain is achieved.

The analysis so far has been based on the coherence length of the crystal as defined by Comley and Garmire.[49] The same arguments can be made in terms of the group delay t_g as defined by Glenn[50] or in terms of the phase matching bandwidth as defined by Miller.[51] The approaches are different but the phenomenon is the same.[48] Here we calculate t_g using the formula[50]

$$t_g = \frac{1}{c}\left[\lambda_L\left(\frac{dn}{d\lambda}\right)_L - \lambda_{SH}\left(\frac{dn}{d\lambda}\right)_{SH}\right] , \qquad (15)$$

where c is the speed of light, and the subscripts L and SH are the fundamental and the second harmonic respectively.

To calculate t_g, $(dn^o/d\lambda)_L$ and $(dn^e(\phi_m)/d\lambda)_{SH}$ are estimated from a table of the ordinary and extraordinary indices of refraction as a function of wavelength at the phase matching angle ϕ_m. Formulas 16 and 17 below from Reference 47:

$$\sin^2 \phi_m = \frac{(n_L^o)^{-2} - (n_{SH}^o)^{-2}}{(n_{SH}^e)^{-2} - (n_{SH}^o)^{-2}} \tag{16}$$

and,

$$\frac{1}{n_e^2(\phi_m)} = \frac{\cos^2 \phi_m}{n_o^2} + \frac{\sin^2 \phi_m}{n_e^2} \tag{17}$$

determine the phase matching angle (Equation 16) and allow calculation of the extraordinary index at phase matching angle as a function of wavelengths (Equation 17). Using data from Zernike[52] on KDP crystals and Formulas 16 and 17, we calculate t_g for KDP is 203 fs/mm for doubling 620 nm pulses.

From this analysis, we conclude that for the generation of 310 nm 50 fs pulses of doubled 620 nm, a KDP crystal thickness no larger than 250 μ must be used which corresponds to M = 1. It is worth mentioning that ADP and BBO are a factor of 1.7 and 2 more dispersive than KDP respectively, thus thickness requirements for these materials are more stringent.

A similar argument can be made in terms of frequency. Phase matching conditions must apply not only at the central wavelength, but at all frequencies of the pulse. A recent theoretical study by Martinez[53] considers the second harmonic generation of broadband femtosecond pulses. After derivation of the phase matching equations, he considers the use of gratings in order to precompensate for the group delay in the crystal so that the group velocity mismatch is cancelled. He proposes an experimental setup for KDP or BBO crystals. In our laboratory, we have used 100 μ thick crystals and up to 0.5 mm crystals when we wanted to increase the conversion efficiency.

6.2 Sum and difference frequency generation

The processes of sum and difference frequency generation[47] involve the nonlinear interaction of two photons of different frequencies ω_1 and ω_2 to produce a new frequency $\omega_3 = \omega_1 + \omega_2$ or $\omega_3 = \omega_1 - \omega_2$, respectively. Similar to SHG, the phase matching conditions are $\Delta k = |k_3 - k_2 - k_1|$ for SFG and $\Delta k = |k_3 + k_2 - k_1|$ for DFG. And similar considerations must be taken into account in calculating the thickness of the mixing crystal.

Sum frequency generation has been used in our laboratory to generate femtosecond pulses at 392 nm by summing the 620 nm femtosecond pulses with the IR YAG pulses at 1064 nm. The conversion efficiency is of approximately 10%. We have also generated femtosecond pulses at 286 nm by summing the femtosecond 620 nm pulses with the second harmonic of the YAG laser at 532 nm. SHG and SFG are the most efficient ways to generate femtosecond pulses in the uv. The output is usually stable with a good quality mode and good polarization.

Difference frequency generation has only been used in our lab for cross correlation processes. By taking pulses at 310 nm and 392 nm we have generated femtosecond pulses at 1480 nm. In this process we were able to characterize the quality of our SHG and SFG pulses and insure they were not being broadened before the experiment. We have also used DFG to characterize pulses of wavelengths between 490 nm and 800 nm by subtracting them from 310 nm. The conversion efficiency of this process is very high and depends primarily on the intensity of the high frequency component, in this case the 310 nm beam.

In Figure 15b, the optical setup for these processes is shown. We have found that focusing any of the high intensity beams can damage the crystals, thus we usually work slightly out of the focus. For SFG and DFG, it is important to match the beam diameters of both incoming beams in addition to minimize the input angle between them. Another important consideration is to know the transmittance of the crystal at the generated frequency. For example, KDP does not transmit light at wavelengths longer than 1000 nm, thus KD*P, which has a longer window in the IR, should be used.

Pulse broadening due to crystal thickness must also be considered for these processes. Tomov et al. have considered several upconversion techniques and their application to subpicosecond pulses.[53] For our particular SFG applications the fundamental or the second harmonic of the YAG laser are essentially CW because of their long wavelength. The group dispersion is therefore more dependent on the difference of the other two wavelengths. For all other cases, one must take the indices of the two extreme wavelengths. Experimentally we have been able to generate 80 fs pulses by SFG.

6.3 Continuum generation

Since the CPM and amplifier setup for the generation of femtosecond pulses is essentially not tunable, the mixing techniques discussed above can only generate a few discrete wavelengths. The ideal source should allow continuous tunability from the uv to the IR without broadening the femtosecond pulses. Smith, Liu and Bloembergen recognized the importance of self-phase-modulation in the presence of self-focusing as a source of superbroadening of short pulses.[54] The bandwidth generated by the phase modulation mechanism increases with decreasing pulse duration, since the nonlinear frequency shift is proportional to the inverse of the pulse duration.[48] Fork et al. observed the generation of a white light continuum from 190-1600 nm as a result of focusing amplified 80 fs pulses into a jet of ethylene glycol.[55] The threshold for continuum generation was found to be 10^{13}-10^{14} W/cm^2 and the efficiency of the process is 100%.

In our laboratory, we use the arrangement shown in Figure 15c for continuum generation. We have found that the focusing lens should be longer than 100 mm focal length and the collimating lens should have a focal length of half that of the focusing lens. This is consistent with the observed factor of two in self-focusing.[55] We have found that very good continuum is achieved by propagating the light through a 1 cm path length quartz cell filled with D_2O. While the continuum extends from the uv to the IR we have been able to use it only from 490 to 800 nm without amplification. The wavelength selection is usually done with interference filters that allow only a slice of the continuum through. We have used interference filters with 10, 5, and 2 nm bandwidths and the resulting pulses have always been less than one hundred femtoseconds in duration.

When the generated continuum is not intense enough, the output must be amplified. The amplification arrangement is shown in Figure 15c. The advantages of amplification is that the beam quality improves, the pulses can be polarized, and the intensity increases by a factor of ~ 100. The amplification process depends on the pump and the dyes that are used. We have used the YAG second harmonic at 532 nm to amplify wavelengths from 550 nm and longer. In order to amplify shorter wavelengths, we would have to use the YAG's third harmonic at 355 nm or the fourth harmonic at 266 nm.

7. The FTS techniques and data acquisition

So far we have described the generation of high intensity femtosecond pulses of almost any wavelength from the uv to the IR. We now describe their use for measuring molecular dynamics of chemical systems. As mentioned before, the femtosecond pulses after amplification and compression are split into two arms of an interferometer. One of the arms, usually the pump, is fixed in length and is subject to the necessary frequency generation process to yield the correct pumping wavelength for the experiment. The other arm, the probe, has a built in computer-controlled delay line and also has an appropriate frequency generation setup to generate the desired probing wavelength. The overall setup is shown schematically in Figure 18 and the experimental procedures are elaborated in the following subsections.

7.1 The delay line

The temporal resolution of the femtosecond experiments depends on two factors, the pulse durations and the precision with which one can control the delay between the pump and probe pulses. The speed of light is ~ 0.3 μ/fs and it is this constant that determines the degree of accuracy needed in the interferometer. There are currently various techniques used in interferometer delay lines that achieve this kind of precision. The best sub-micron control is provided with piezoelectric transducers, (PZT), that change their size according to the applied voltage. These devices can have nanometer and even sub-nanometer resolution. Their disadvantages are that they do not respond linearly to voltage and that they cannot travel longer than a fraction of a millimeter (about 6 ps in a delay line). These devices have been used when one wants to scan a delay at a very high speed. Usually one can scan at 20 to 30 Hz and thus achieve a real time acquisition,provided the repetition of the laser system is several kHz. A similar device is a magnetic coil similar to the one used for speakers. Once the displacement as a function of voltage is calibrated, these devices are also designed to oscillate at very high rates. Magnetic coils are not as precise as PZTs but they are much less expensive.

In our setup, we have used a PZT in order to take real time autocorrelations of the CPM laser, but most of our delay control is done with an optically encoded micropositioner (also called actuator). The micropositioner has an optically encoded stepping motor, a gear box and a micrometer screw. Each step is approximately 60 nm, which corresponds to a 0.4 fs delay. The advantages of these micropositioners is that, in principle, they can travel any distance limited only by the length of the translation stage. The motion is very fast and linear. The controller for the micropositioners is interfaced with the computer, and automation of an experimental run is simple.

7.2 The experimental setup

Almost every experiment in our laboratory has required a different experimental setup. Besides the changes in wavelengths, pulse durations and intensities, the chemical processes have required different experimental cells. The design considerations for these experimental cells usually are based on the requirement that femtosecond pulses must not be broadened.

Most of the experiments measure laser induced fluorescence, LIF, as the signal; in some cases we have measured absorption, enhanced emission and ionization. These experiments are performed in the gas phase and usually under low pressures. For these purposes, we have used three different kinds of cells: high- and low-temperature flow cells, and static cells. The flow cells allow the continuous renovation of the sample and in some cases we have used them for two kinds of experiments by pumping away one sample and feeding in the new one. Having to connect continuously with a vacuum line makes the use of these cells more tedious than the static cells.

When the continuous renovation of the sample is not needed, we have made sealed quartz cells. These cells are prepared under vacuum and they can be heated to very high temperatures according to the experimental needs. Since no connection to vacuum in necessary, these cells are very compact and easy to use. They are made of 1/16" quartz which causes minimal pulse broadening.

Other types of experimental cells have included liquid flow cells and high pressure cells for experiments in up to 100 atm. In all the designs, the window thickness was calculated to be minimal and quartz was used because of its low GVD. Future experiments will also be done in a molecular beam designed specifically for the femtosecond setup.

7.3 Data acquisition

The data acquisition process consists in collecting the measured signal, averaging it in a Boxcar integrator, and feeding it to the computer for each delay value. In our laboratory, the entire data acquisition is carried out by a Macintosh II computer which controls the delay line and inputs the signal levels from the Boxcar. The data acquisition program was primarily written by Mark Rosker with enhancements by Lawrence Peng and Marcos Dantus.

The data acquisition program, included in Appendix 2, can be divided into six categories (see Figure 19): (1) data collection procedures, (2) communications, (3) file management, (4) nonlinear least-square fitting procedures, (5) the graphic interface, and (6) the header files. Each of these sections is described below.

7.3.1 Data acquisition procedures

The data acquisition project provides the user with a graphics and data windows. The handling of the different procedures of the program is governed by Data Acquisition.c. The actual data collection is dictated by the program Collection800.c. Before running the data collection program, one specifies the parameters such as the number of scans, the number of data points per scan, and the starting position, for example. These numbers are entered by making use of the parameters dialog box shown in Figure 20. Once the parameters are entered, the user can save them and use them for all similar runs.

The Collection800.c program directs the acquisition procedure step-by-step according to the parameters entered. Every point acquired is displayed on the screen. Once a scan finishes, the values are summed to those of previous scans and the resulting array is plotted and saved to disk in a temporary file. When the complete run is over, the results are stored in a permanent file.

7.3.2 Communications

In order to acquire data, the computer has to communicate with the micropositioner (actuator), controller and the boxcar. The communication protocol is different for both cases because the actuator controller has a serial interface and the boxcar has a parallel interface. Serial communications are performed by the program Serial Comm.c and actuator specific commands are handled by the routines in Actuator.c. The computer has two serial input/output ports for communications, these are the printer and modem ports. Currently, the modem port is used for serial communications with the actuator controller. The communication parameters are specified by making use of the communications dialog box shown in Figure 21.

Communications with the boxcar are routed through an IEEE488 board from National Instruments (NB-GPIB). The configuration procedure of the board is included in Appendix 3. National Instruments supplies the program Li.c which includes all the commands that one can give the board to perform. The boxcar commands such as GetBoxcarVoltage are included in NewBoxcar.c and they consist of combinations of statements and routines in Li.c. The boxcar has an interface module which allows 8 analog input/output BNC ports. Port #3, for example, is currently being used to interface the computer with the monochromator by checking if the scanning pin is either high or low. Similarly, the computer can get the input voltage on any of the analog ports or, conversely, it can drive any of the ports to a specified voltage.

7.3.3 File management

The data acquisition program saves every scan to a temporary file called "Most Current Scan" as a safeguard in case of a malfunction. Once the run is over the scan is saved in a permanent file. The program can also read a data set. These read and write procedures are managed by the following routines: Decode.d, file.mini.c, Write.c, ReadSaveFile.c and Utilities.c.

Saving of the data is done by Write.c except for the temporary storage which is handled by ReadSaveFile.c. Reading an old file is accomplished by Decode.c. The program file.mini.c allows limited interaction with the file such as saving and closing a file. The program Utilities.c is mainly a collection of useful routines that make the programing task simpler.

7.3.4 NLS fitting

Once a data set is acquired, the data acquisition program includes some data analysis routines based on the nonlinear least-squares fitting algorithm. The engine of the fitting routine is based on five routines provided by the Numerical Recipes book[56] one driving program (MickeyMousefit.NLS.c) and one file containing the functional forms to be fitted. Currently the most-used functions are the Gaussian, Lorentzian and the hyperbolic-secant-squared. The driving routine estimates the starting parameters for the fit and the fitting is executed. The fitting dialog box shown in Figure 22 allows the user to modify the starting parameters or to chose a section of the file to be fitted. The selection is accomplished by pointing the mouse to the initial point and dragging it to the final point of the fit.

The NLS fitting routines are regularly used for the determination of the autocorrelations and the spectral widths. The quality of the fit to Gaussian or to hyperbolic-secant-squared functions allows us to estimate the distortion in our pulses. Other functions that are very useful are the line, and the single- and bi-exponentials.

7.3.5 Graphic interface

The program follows the graphic interface consistent with the Macintosh computers. Several routines are involved in the process. For example, mini.windows.c takes care of the graphics and data windows, dialogRoutines.c takes care of all dialog boxes used in the data acquisition program. The routines plot.c and scale.c take care of plotting the data on the graphics window. The Fit.Dlog.c routine takes care of the fitting parameters entered in the fitting dialog box. MousePaint.c gives the user a visual feedback when selecting a portion of the data to be fitted by shading the region in real-time as the user dregs the mouse pointer. Finally, the procedure pleasewait.c changes the cursor from an arrow to a watch during data acquisition.

7.3.6 Header files

The program uses a very large number of variables and routines which interact with different sections. In order to keep track of all these variables, and to make sure that they are all defined for the compiler, header files are constructed. The DataAcquisition.h header contains definitions related to the data acquisition procedure such as the maximum number of data points permissible, the meaning of every entry in the parameters dialog box, and the port that is being used for serial communications. The IEEE488 communications requires four header files in order to reduce all the complicated control language of the board into simple commands like write and read. These headers are: DrInterface.h, GPIBres.h, IEEE.h, and sys.h.

The least squares program has the header least_squares.h that defines the maximum number of parameters to be fitted. Finally, the fitting routine makes use of NRUTIL.H which is a header provided for the Numerical Recipes routines.

References

1. H. Mocker and R. Collins, Appl. Phys. Lett. 7, 270 (1965).
2. A. J. DeMaria, D. A. Stetser and H. Heyman, Appl. Phys. Lett. 8, 22 (1966).
3. W. Schmidt and F. P. Schafer, Phys. Lett. 26A, 558 (1968).
4. E. P. Ippen, C. V. Shank and A. Dienes, Appl. Phys. Lett. 21, 348 (1972).
5. R. L. Fork, B. I. Green and C. V. Shank, Appl. Phys. Lett. 38, 671 (1981).
6. C. V. Shank, R. L. Fork, R. Yen, R. H. Stolen and W. J. Tomlinson, Appl. Phys. Lett. 40, 761 (1982).
7. R. L. Fork, C. H. Brito Cruz, P. C. Becker and C. V. Shank, Opt. Lett. 12, 483 (1986).
8. In the Ultrafast Phenomena VI (1990) meeting most of the new femtosecond generation techniques involved modelocked solid state lasers such as YAG and YLF glasses, alexandrite, Ti sapphire and color center lasers.
9. J. A. Valdmanis, R. L. Fork and J. P. Gordon, Opt. Lett. 10, 131 (1985); J. A. Valdmanis and R. L. Fork, IEEE J. Quantum Electron. QE22, 112 (1986) for a more recent analysis see H. Avramopoulos, P. M. W. French, G. H. C. New, M. M. Opalinska, J. R. Taylor, and J. A. R. Williams, IEEE J. Quantum. Electron. QE25, 2469 (1989).
10. M. Maeda, Laser Dyes, (Acad. Press, Tokyo, 1984).
11. A. E. Siegman, Lasers, (University Science Books, CA, 1986).
12. B. Couillaud and V. Fossati-Bellani, Lasers and Applications, 79 (January 1985).
13. C. V. Shank, R. L. Fork and F. Beisser, Laser Focus June 59 (1983).
14. A simple explanation of the origin of group velocity dispersion is given by: S. Solimeno, B. Crosignani and P. Di Porto, Guiding Diffraction and Confinement of Optical Radiation, (Acad. Press, Orlando 1986).

15. S. De Silvestri, P. Laporta and O. Svelto, Opt. Lett. 9, 335 (1984); W. Dietel, E. Dopel, K. Hehl, W. Rudolph and E. Schmidt, Opt. Commun. 50, 179 (1984); A. M. Weiner, J. G. Fujimoto and E. P. Ippen, Opt. Lett. 10, 71 (1985); P. Laporta and V. Magni, Appl. Opt. 24, 2014 (1985); D. N. Christodoulides, E. Bourkoff, R. I. Joseph and T. Simos, IEEE J. Quantum. Electron. QE22, 186 (1986) and references therein.
16. S. De Silvestri, P. Laporta and O. Svelto, IEEE J. Quantum. Electron. QE20, 533 (1984).
17. A. M. Johnson and W. M. Simpson, Opt. Soc. Am. B2, 619 (1985).
18. O. E. Martinez, R. L. Fork and J. P. Gordon, Opt. Lett. 9, 156 (1984).
19. J. -C. Diels, W. Dietel, J. J. Fontaine, W. Rudolph and B. Wilhemi, J. Opt. Soc. Am. B2, 680 (1985).
20. F. Salin, P. Grangier, G. Roger and A. Brun, Phys. Rev. Lett. 56, 1132 (1986).
21. L. F. Mollenauer and R. H. Stolen, Fiberoptic Technology April, 193 (1982).
22. N. J. Frigo, IEEE J. Quantum. Electron. QE19, 511 (1983).
23. A. Yariv, Quantum Electronics, 2nd ed. (J. Wiley, New York, 1975).
24. J. M. Drake, R. I. Morse, R. N. Steppel and D. Young. Chem. Phys. Lett. 35, 181 (1975).
25. A. Migus, C. V. Shank, E. P. Ippen and R. L. Fork, IEEE J. Quantum Electron. QE18, 101 (1982).
26. A. Migus, J. L. Martin, R. Astier and A. Orzag, in Picosecond Phenomena II, Springer Series in Chemical Physics Vol 14, R. Hochstrasser, W. Kaiser, and C. V. Shank, Eds. (Berlin, Springer, 1980).
27. R. L. Fork, C. V. Shank and R. T. Yen, Appl. Phys. Let. 41, 145 (1982).
28. A. J. Alcock, C. De Michelis, V. V. Korobkin and M. C. Richardson, Appl. Phys. Lett. 14 145 (1969).
29. B. B. Snavely, in Topics in Applied Physics Vol I, Dye Lasers, F. P. Schafer, ed, (Springer-Verlag, N. Y., 1973).

30. C. V. Shank, in Ultrashort Laser Pulses, W. Kaiser ed., (Springer-Verlag, Heidelberg 1988).
31. F. Gires and Tournois, C. R. Acad. Sci Paris 258, 6112 (1964).
32. J. A. Giordmaine, M. A. Duguay, J. W. Hansen, IEEE J. Quantum Electron. QE4, 252 (1968).
33. E. B. Treacy, IEEE J. Quantum Electron. QE5, 454 (1969).
34. R. A. Fischer, P. L. Kelly, T. K. Gustafson, Appl. Phys. Lett. 14, 140 (1969).
35. L. F. Mollenauer, R. H. Stolen and J. P Gordon, Phys. Rev. Lett. 45, 1095 (1980).
36. W. H. Knox, R. L. Fork, M. C. Downer, R. H. Stolen, C. V. Shank and J. Valdmanis, Appl. Phys. Lett. 46, 1120 (1985).
37. J. D. Jackson, Classical Electrodynamics, 2nd ed. (J. Wiley, N. Y., 1975).
38. F. Salin and A. Brun, J. Appl. Phys. 61, 4736 (1987).
39. Z. Bor and B. Rácz, Appl. Opt. 24 (1985).
40. R. L. Fork, O. E. Martinez and J. P. Gordon, Opt. Lett. 9, 150 (1984); J. D. Kafka and T. Baer, Opt. Lett. 2, 401 (1987).
41. Schott Optisches Glas 3111d, Catalog of the Schott Optical Glass Inc., Mainz, Federal Republic of Germany (1980).
42. M. Maier, W. Kaiser and J. A. Giordmaine, Phys. Rev. Lett. 17, 1275 (1966); Phys. Rev. 177, 580 (1969).
43. J. C. M. Diels, J. L. Fontaine, I. C. McMichael and F. Simoni, Appl. Opt. 24, 1270 (1985).
44. G. R. Flemming, Chemical applications of ultrafast spectroscopy, (Oxford University Press, N. Y. 1986); similar tables have been compiled before by: R. C. Greenhow and A. J. Schmidt, Advan. Quantum Electron. 2, 157 (1974); K. L. Sala,G. A. Kenney-Wallace, and G. E. Hall, IEEE J. Quantum Electron. QE16, 990 (1980).
45. T. Kurobori, Y. Cho and Y. Matsuo, Opt. Commun. 40, 156 (1981).
46. P. Franken, A. Hill, C. Peters and G. Weinreich, Phys. Rev. Lett. 7, 118 (1961).
47. A. Yariv, Quantum Electronics, 2nd ed. (J. Wiley, N. Y. 1975).

48. A. Laubereau, in Ultrashort Laser Pulses, W. Kaiser ed., (Springer-Verlag, Heidelberg 1988).

49. J. Comly and E. Garmire, Appl. Phys. Lett. 12, 7 (1968).

50. W. H. Glenn, IEEE J. Quantum Electron. QE5, 284 (1969).

51. R. C. Miller, Phys. Lett. 26A, 177 (1968).

52. F. Zernike Jr., J. Opt. Soc. Am. 54, 1215 (1964).

53. O. E. Martínez, IEEE J. Quantum Electron. QE25, 2464 (1989).

54. I. V. Tomov, R. Fedosejevs and A. A. Offenberger, IEEE J. Quantum Electron. QE18, 2048 (1982).

55. W. L. Smith, P. Liu and N. Bloembergen, Phys. Rev. A15, 2396 (1977).

56. R. L. Fork, C. V. Shank, C. Hirliman and R. T. Yen, Opt. Lett. 8, 1 (1983).

57. W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, Numerical Recipes, (Cambridge University Press, N. Y. 1986).

TABLE I: Values of the index n_0 and the parameters $n_0' = (dn/d\lambda)_{\lambda_0}$ and $n_0'' = (d^2n/d\lambda^2)_{\lambda_0}$ for different media at 620 nm.

Medium	n_0	n_0' (μm^{-1})	n_0'' (μm^{-2})
FeD 05–25	1.8011	−0.13326	0.67787
SiO_2	1.4572	−0.02984	0.12144
BK7	1.5159	−0.03756	0.16641
Water	1.33	−0.02729	0.1303

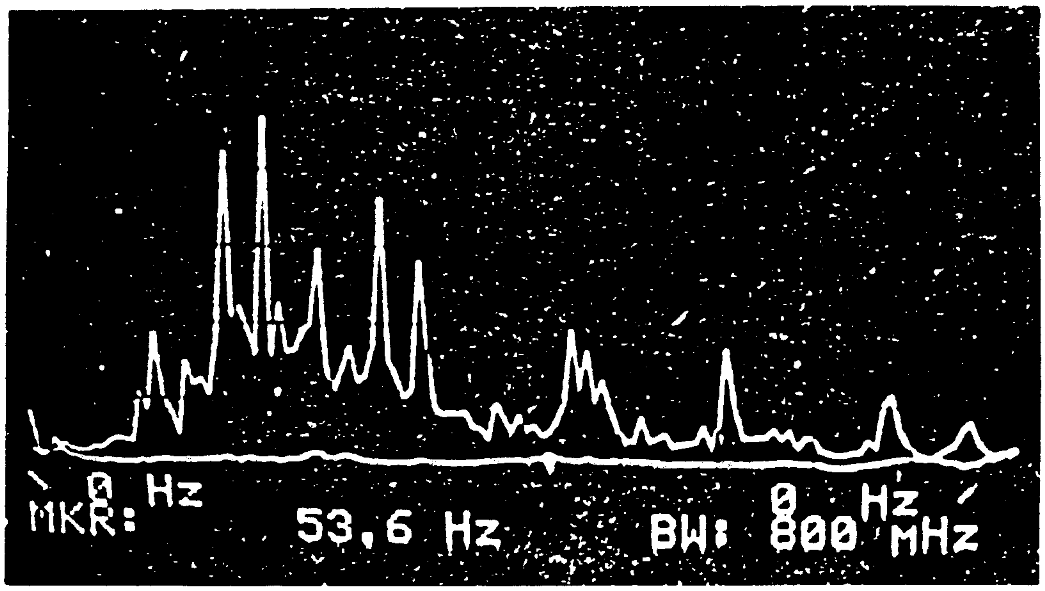


Figure 1 Seismic measurement taken on the surface of the optical bench for two conditions. The noisy spectrum corresponds to the table before vibrational isolation. The quiet spectrum was taken with the table floating on vibration dampening pistons.

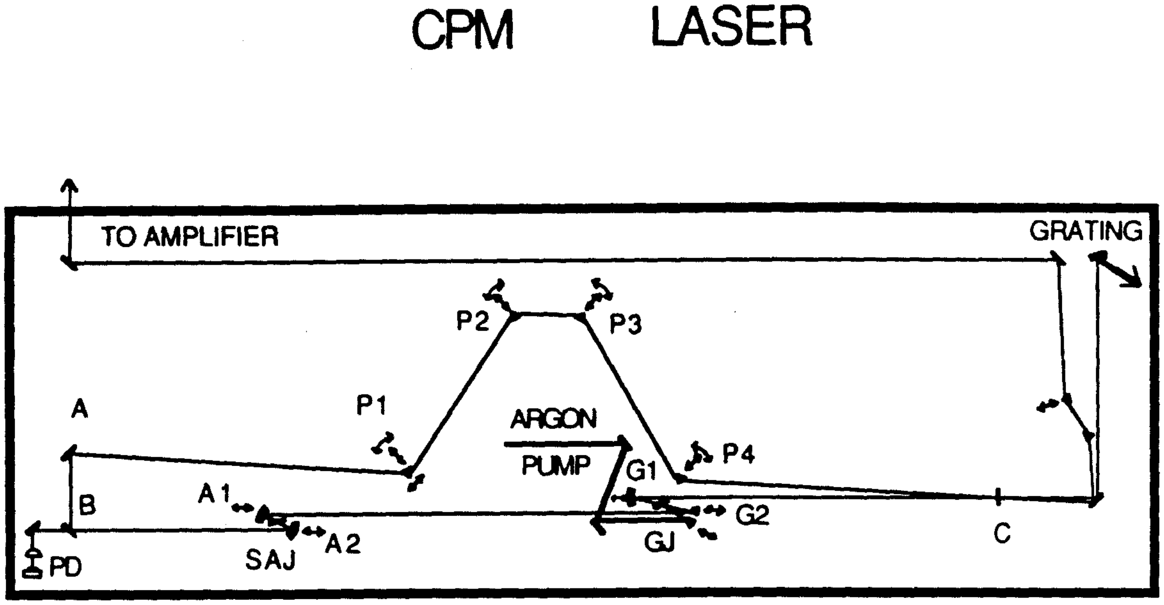
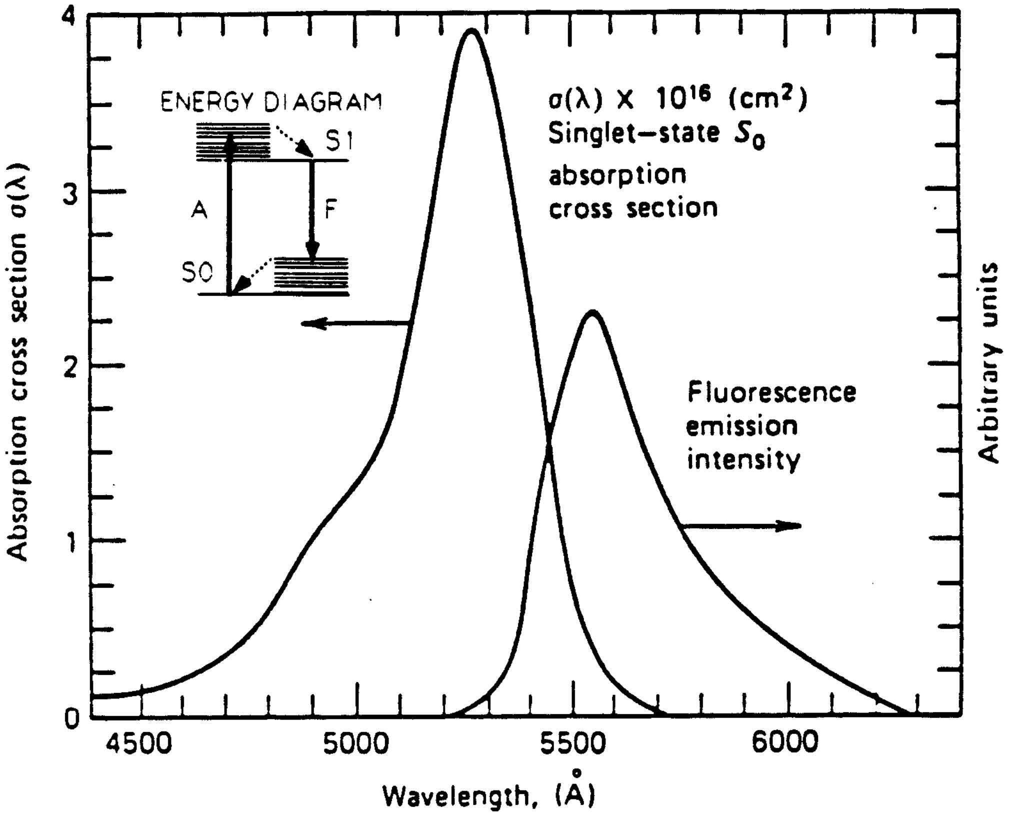
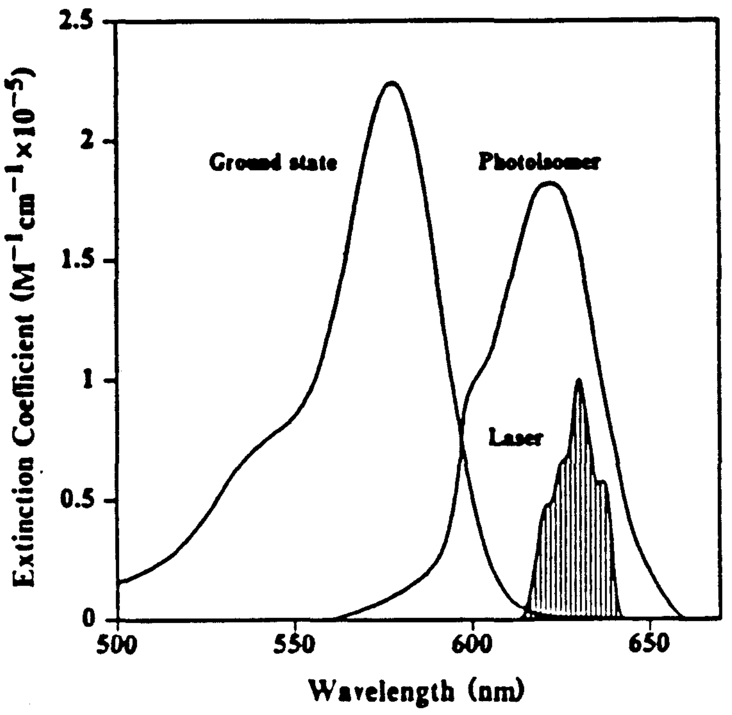


Figure 2 Schematic drawing of the CPM laser. The laser cavity consists of seven mirrors: A, B, C, G1, G2, A1 and A2. The gain jet is GJ and the saturable absorber jet is SAJ. There is a four prism arrangement, P1, P2, P3 and P4. A < 1% leak through mirror B is detected by the photodiode PD for synchronization with the amplifier. Mirror C is 2% transmissive and serves as the output coupler. Notice that the angle ACB is very narrow. This is done on purpose, so that mirrors A and B can be coated for 45^o reflection and the output coupler for normal incidence.



Singlet-state absorption and fluorescence spectra of rhodamine 6G obtained from measurements with a 10^{-4} molar ethanol solution of the dye.

Figure 3 Singlet-state absorption and fluorescence spectra of rhodamine 6G obtained from measurements with a 10(-4) molar ethanol solution of the dye. The insert in the upper left corner shows a schematic diagram for the absorption (A) and fluorescence (F) between the S_0 and S_1 singlet states. The spectra are taken from reference 23.



Laser spectrum of 34 fs pulses (see Fig. 3) in relation to the absorption spectra of the DODCI ground and photoisomer states.

Figure 4 Laser spectrum of 34 fs pulses in relation to the absorption spectra of the DODCI ground and photoisomer states. The spectra are taken from reference 9.

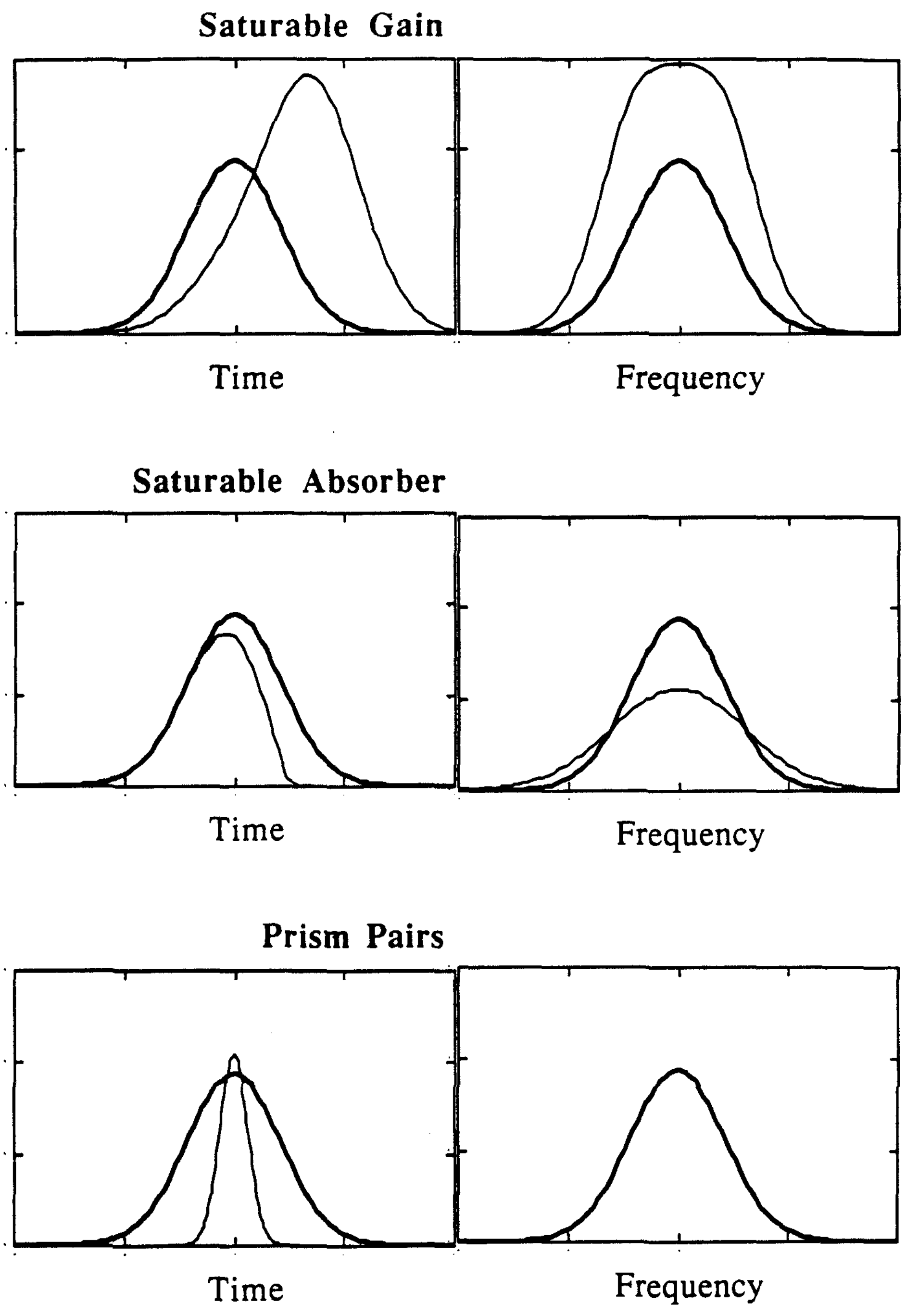


Figure 5 Pulse shaping effects that take place during the operation of a CPM laser (heavy line - before; light line - after). Propagation through a saturable gain preferentially amplifies the leading edge of the pulse and broadens the pulse spectrally. Propagation through a saturable absorber preferentially attenuates the leading edge of the pulse (the opposite of the saturable gain) and the pulse broadens spectrally due to self phase modulation. Propagation through the prism pairs arrangement compresses the pulse duration, leaving the spectrum intact.

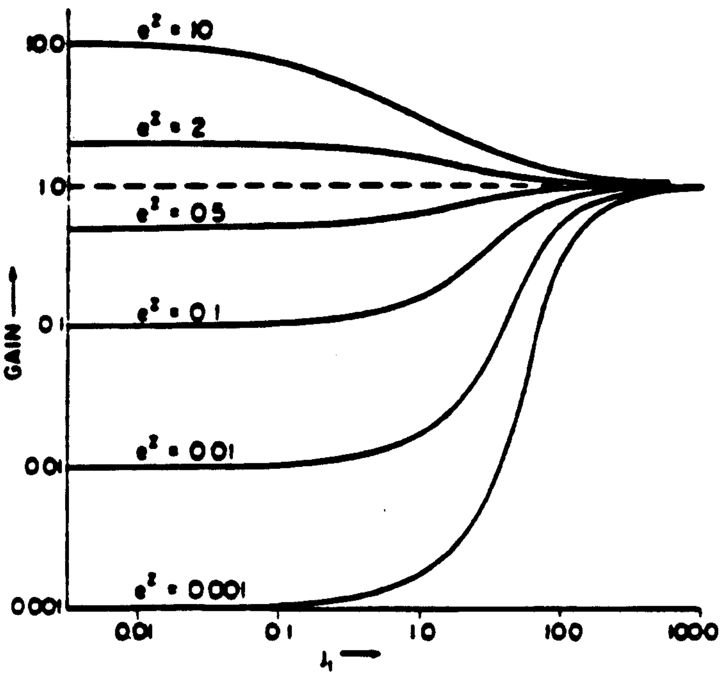


Figure 6 Overall gain as a function of initial energy (j_i) for different optical thicknesses (e^z). Note that for low initial energies, Beer's law is valid, but for
$j_i > 1$, saturation begins to lower the expected gain. Figure taken from reference 22.

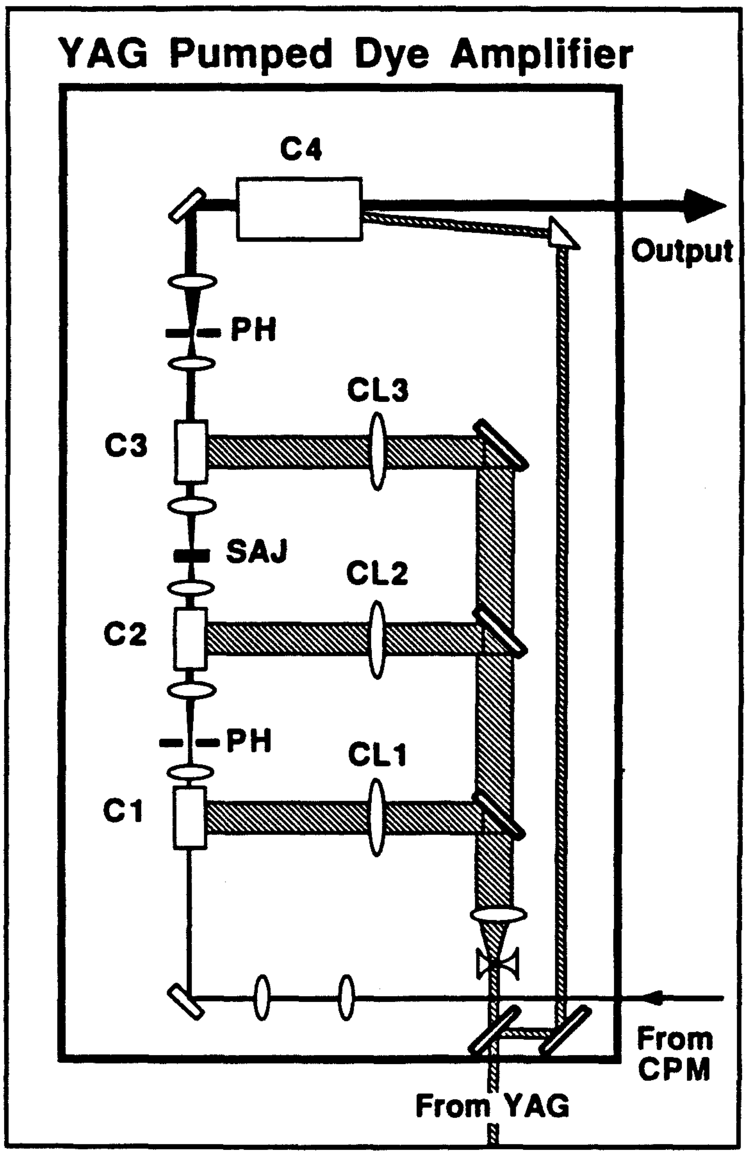
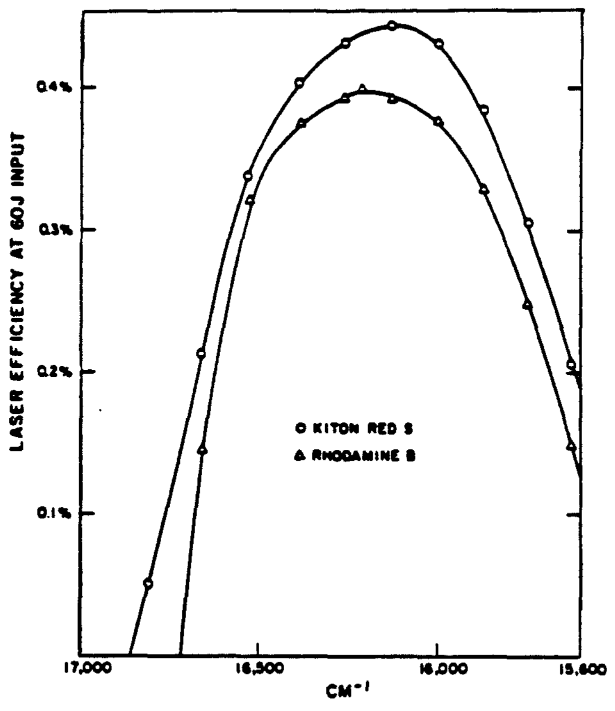
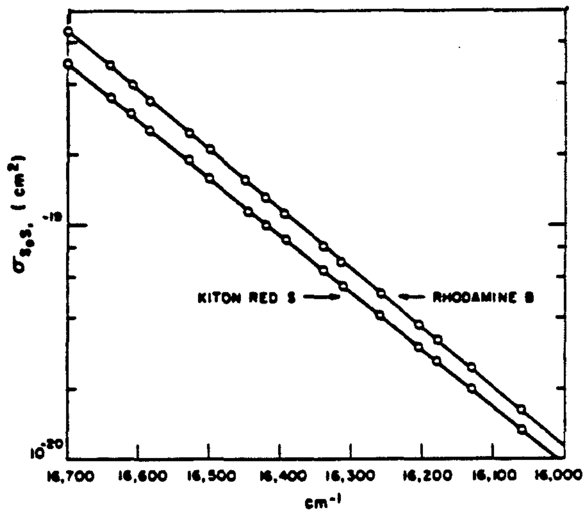


Figure 7 Schematic drawing of the four stage amplifier. C1, C2 C3 and C4 are the four dye cells (stages) of the amplifier. Along the beam path there is a telescope and then two pinholes (PH) and one saturable absorber jet (SAJ), each between a 2X telescope. The YAG laser is divided by a beam splitter. 70% is reflected towards the fourth stage and the rest is transmitted and expanded in a telescope. The expanded beams are focused by cylindrical lenses CL1, CL2 and CL3.



Laser tuning curves for 1 X 10^{-4} M kiton red S and rhodamine B in ethanol pumped with a coaxial flashlamp.



Self-absorption cross section $\sigma_{S_0-S_1}$ for kiton red S and rhodamine B in absolute ethanol. ($\sigma_{S_0-S_1}$ must be multiplied by 2.3.)

Figure 8 Laser emission and self absorption. For Kiton red and rhodamine B (sulforhodamine 640). Notice the lower absorption cross section of Kiton red. (The curves are taken from reference 24).

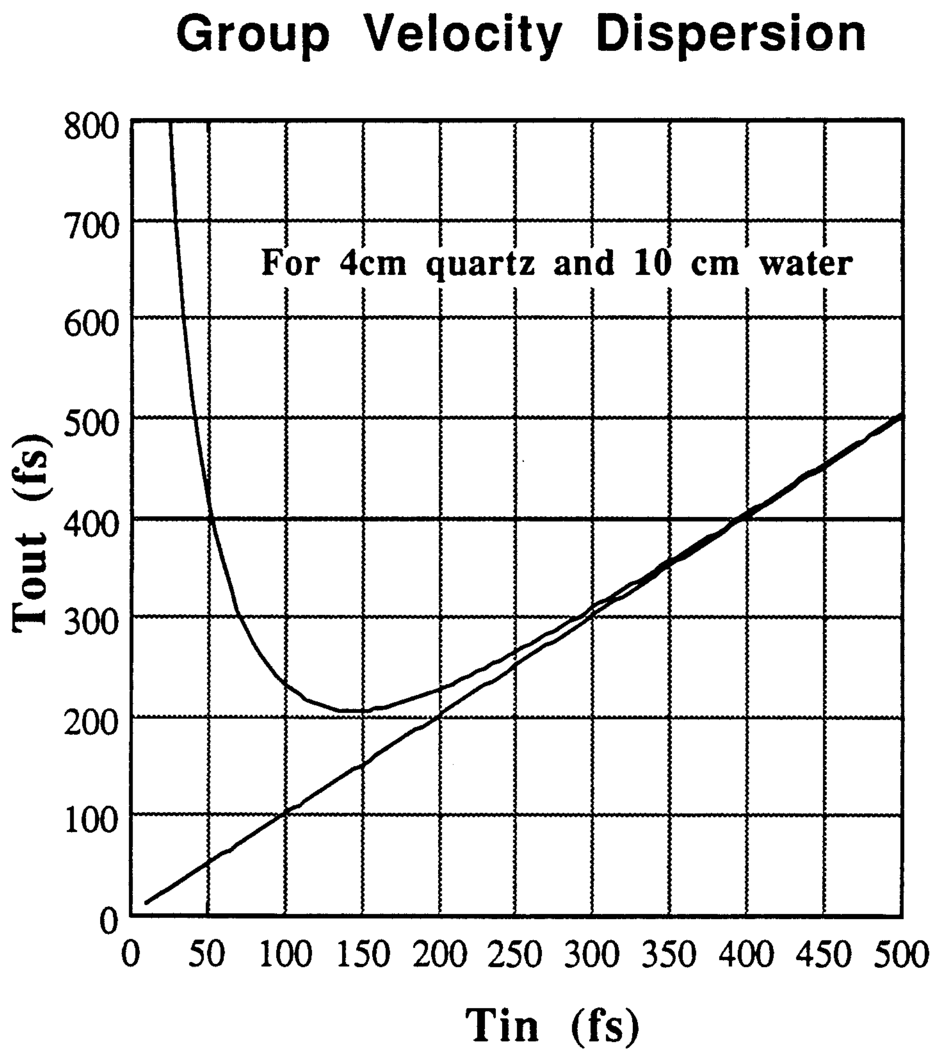
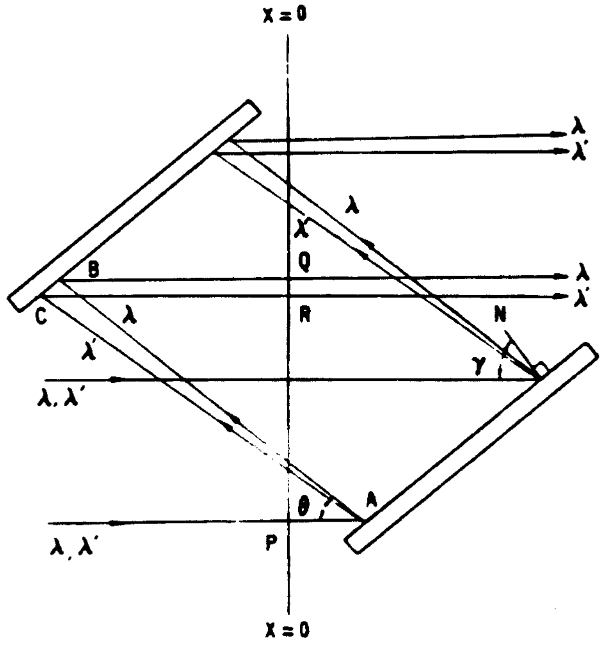
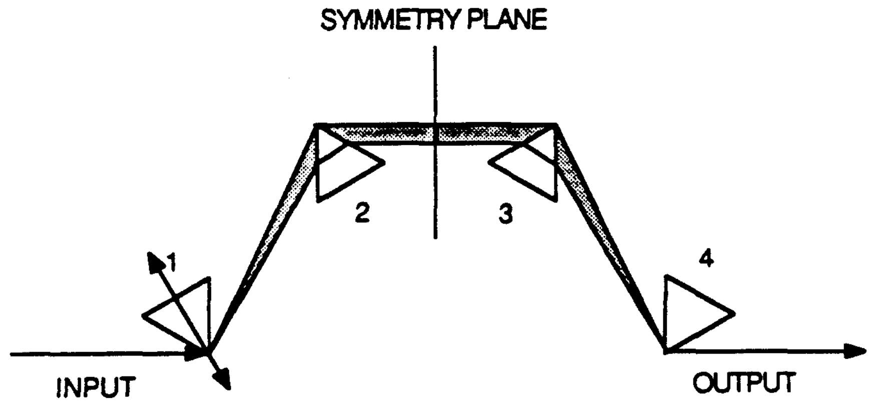


Figure 9 Group velocity dispersion calculated for an equivalent amplifier setup with 4 cm of quartz and 10 cm of water. The calculation is made using Equation 9 and the parameters in Table 1. The straight line corresponds to no dispersion.

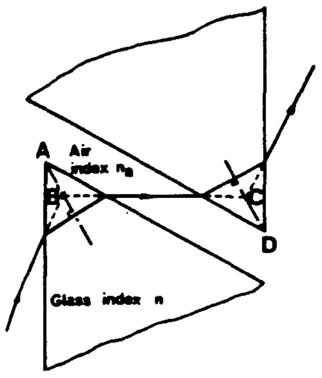


Geometrical arrangement of diffraction gratings used for pulse compression, etc. The angle of incidence is γ, and θ is the acute angle between the incident and diffracted rays. The ray paths are shown for two wavelength components with $\lambda' > \lambda$. Since the path length for λ' is greater than that for λ, longer wavelength components experience a greater group delay.

Figure 10 Optical arrangement of the grating pair as originally proposed by Treacy[33] for compression of light pulses. Notice that the shorter wavelengths λ have a shorter path length than the longer wavelengths λ'.



PRISM PAIRS ARRANGEMENT



Geometrical construction to obtain the optical path for two prisms.

Note that $L = BC$ and $e = AB + CD$.

Figure 11 Schematic drawing of the prism pairs arrangement. Notice that translation of prism #1 (or any other prism) along the bisecting angle does not affect the direction of the output and is used to regulate the amount of negative or positive dispersion of the arrangement. If the prisms are oriented properly, the input and output beams are collinear and the beam diameter is not affected. The insert shows the how to obtain the optical path of the beam along the glass (e) and in air (L). These parameters are then used in Equation 12 to calculate the dispersion of the prism arrangement.

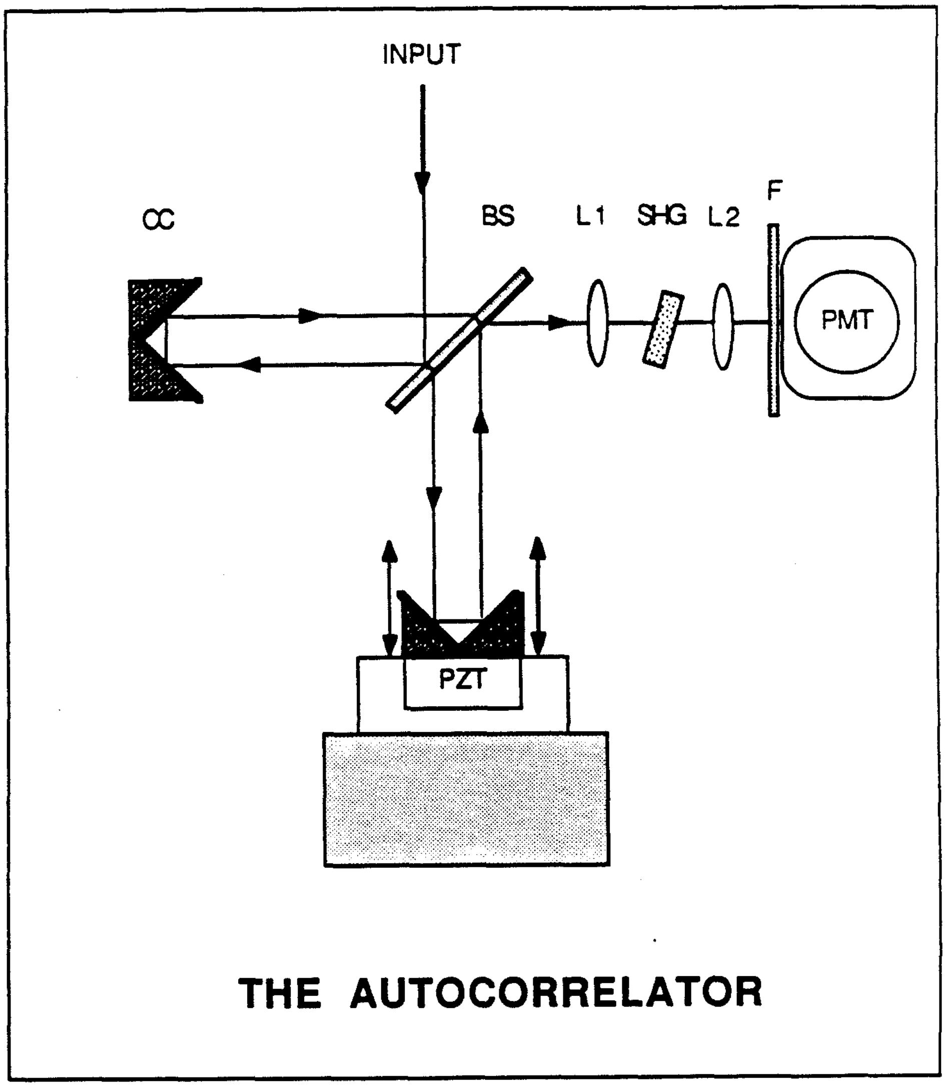


Figure 12 Schematic drawing of the autocorrelator. It consists of a very thin (0.2 mm) beamsplitter that divides the beam into two arms. One arm is directed towards a static corner cube (CC) and the other to a corner cube mounted on a piezoelectric transducer (PZT) or an actuator. The combined beams are focused by lens L1 into a second harmonic generation (SHG) crystal and are collimated by lens L2. An optical filter (F) rejects the fundamental and passes the second harmonic light that is detected by a photomultiplier tube (PMT). Note that both beams go through the same amount of glass. This arrangement can be collinear or non-collinear.

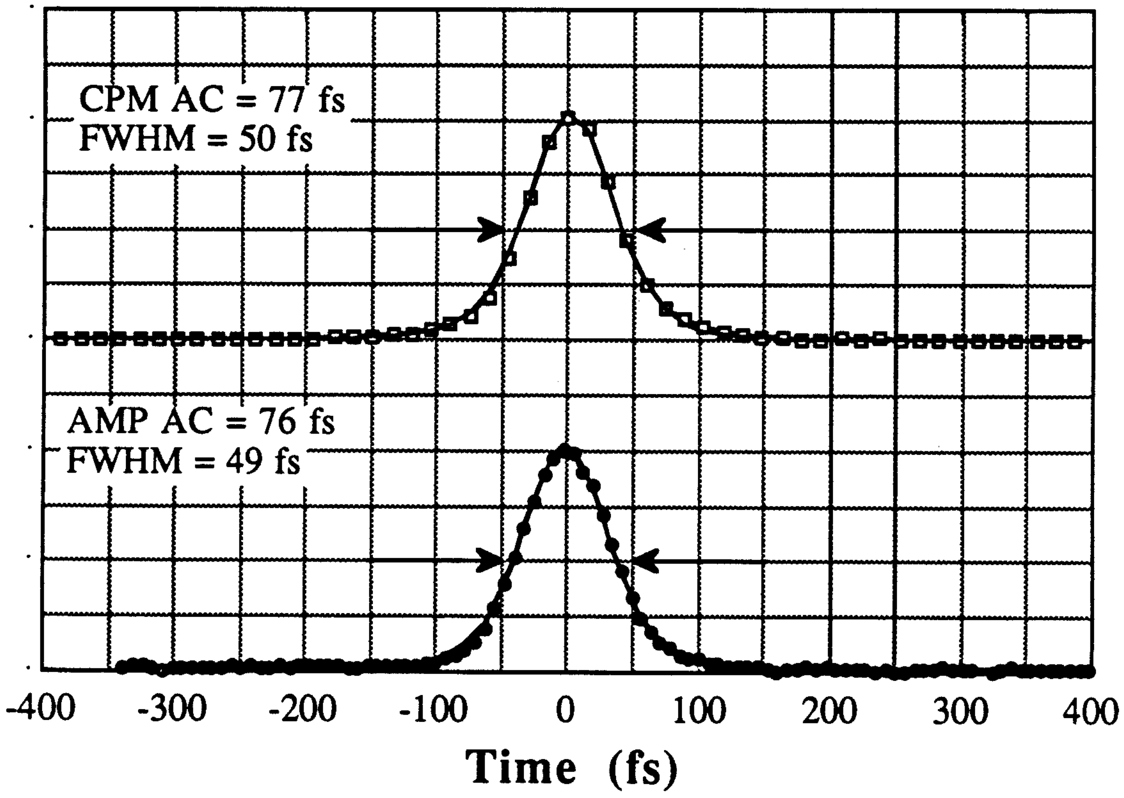


Figure 13 Intensity autocorrelation of the CPM output pulses, and intensity autocorrelation of the amplified pulses after pulse compression.

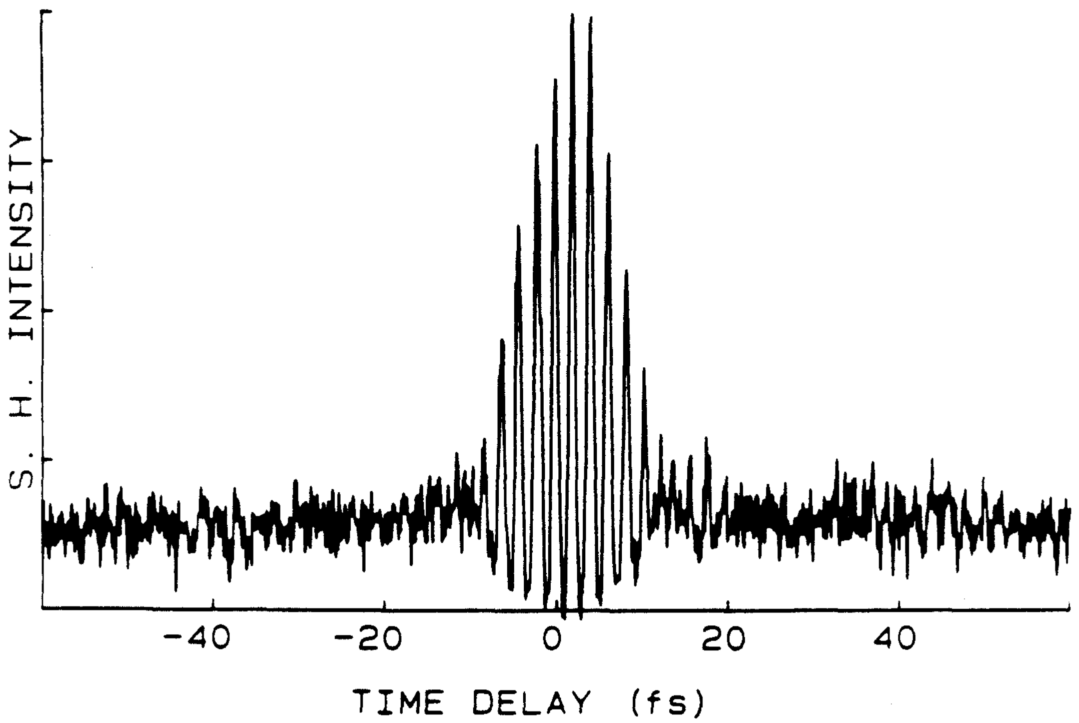


Figure 14 Plot of the interferometric autocorrelation function measured for a 6 fs optical pulse. The figure is taken from Reference 7.

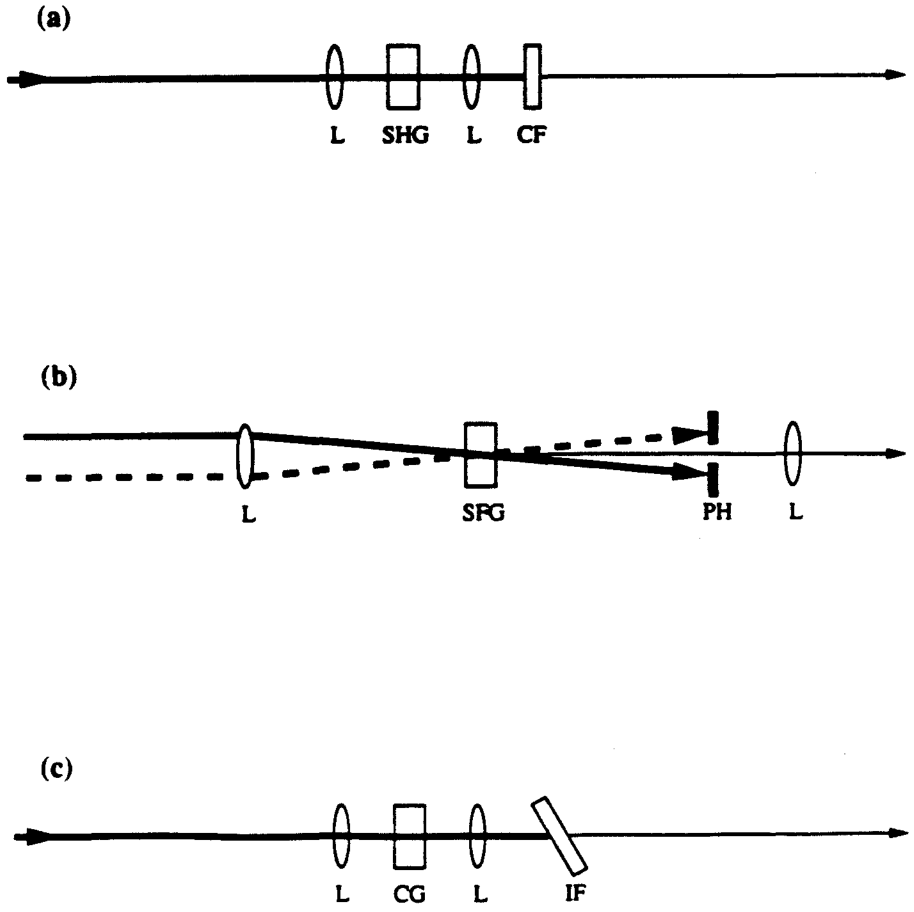


Figure 15 Optical arrangements used for (a) second harmonic generation, (b) sum and difference frequency generation and (c) for continuum generation. CF stands for color filter, PH is an iris or pinhole, CG is the continuum generation cell, and IF is an interference filter used to select a portion of the continuum spectrum. Figure adapted from Rose et al., J. Chem. Phys. 91, 7415 (1989).

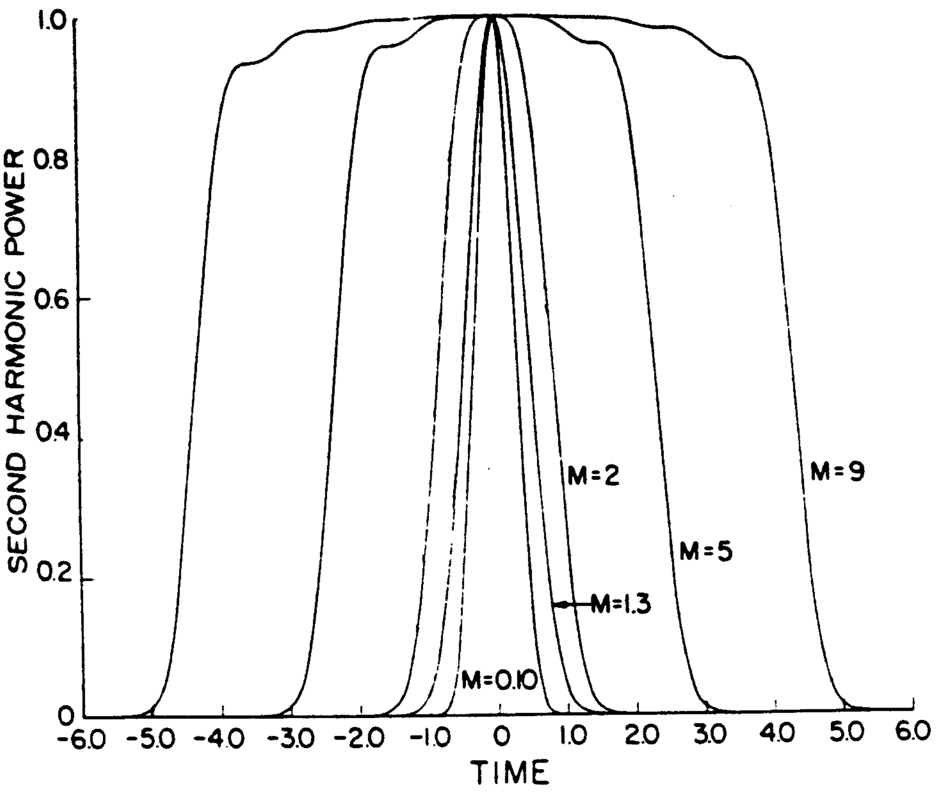


Figure 16 Second harmonic pulse width as a function of the unitless crystal thickness parameter (M). All pulses are normalized to the same peak power. Figure taken form Reference 49.

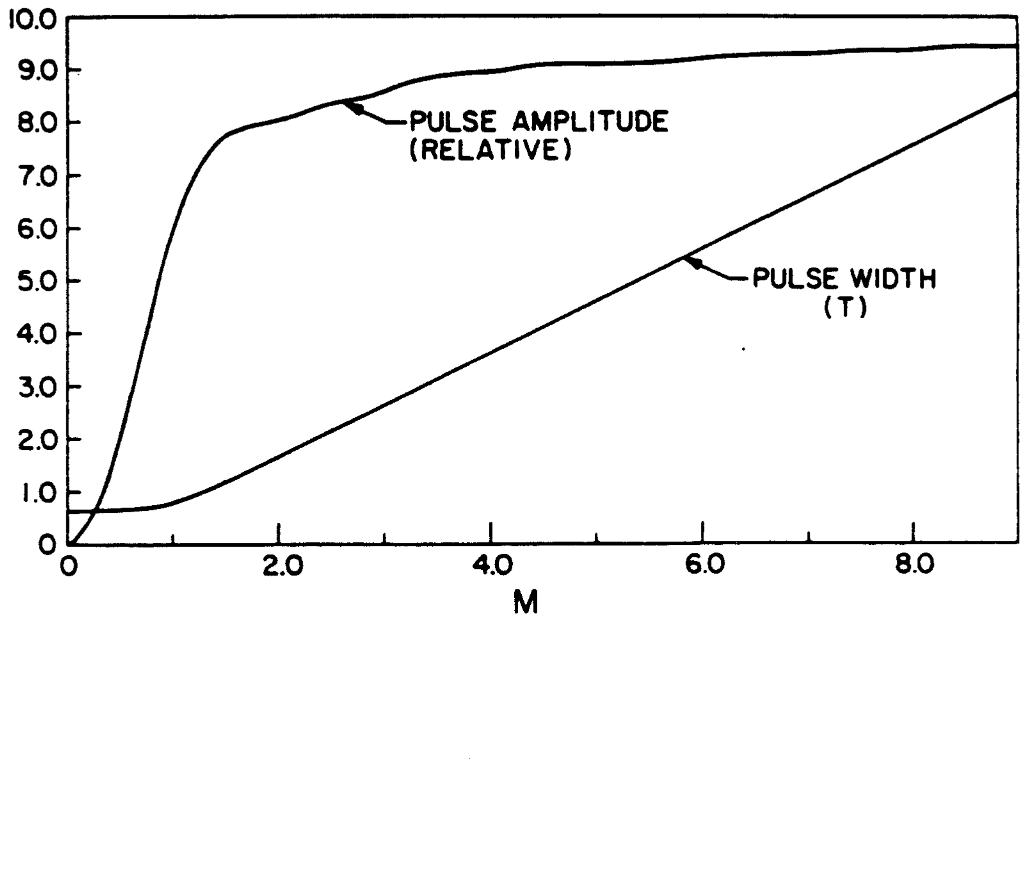


Figure 17 Plot of the pulse amplitude and the pulse width as a function of the crystal thickness M. Notice that for $M < 1$, the pulse width remains unaffected while the pulse amplitude increases as the square of the thickness. For $M > 2$, the pulse width increases linearly with the crystal thickness while the amplitude remains fairly constant. Figure taken from Reference 49.

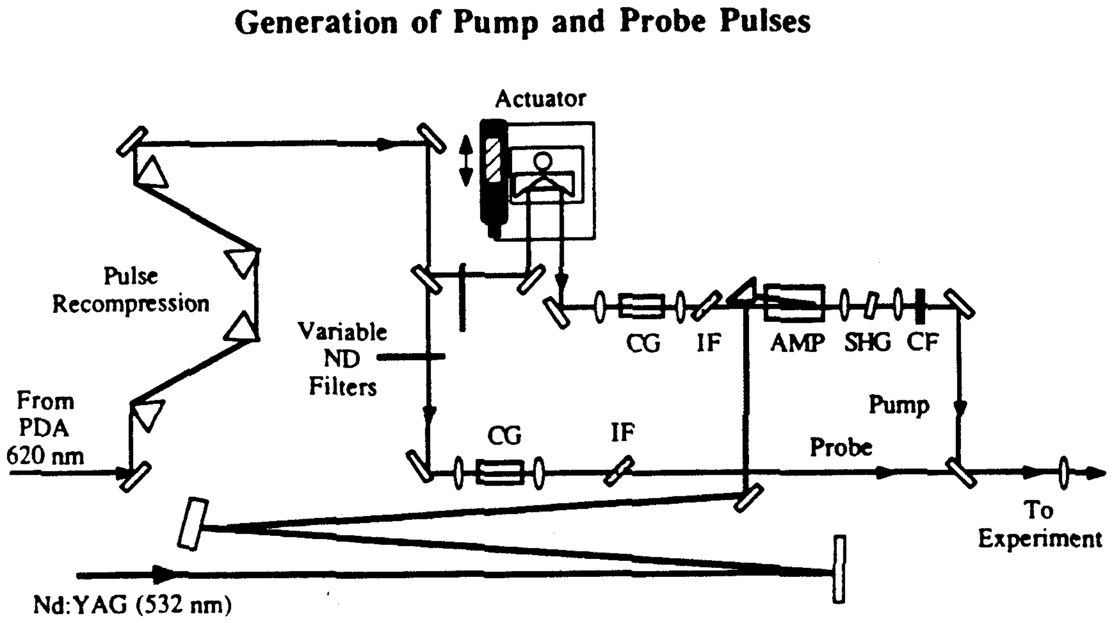


Figure 18 Schematic drawing showing the optical arrangement for the generation of the pump and probe beams. Key: CG = continuum generation; IF = interference filter; SHG = second harmonic generation crystal; CF = color filter; AMP = amplification stage; ND = neutral density filter. Figure adapted from Rose et al., J. Chem. Phys. 91, 7415 (1989).

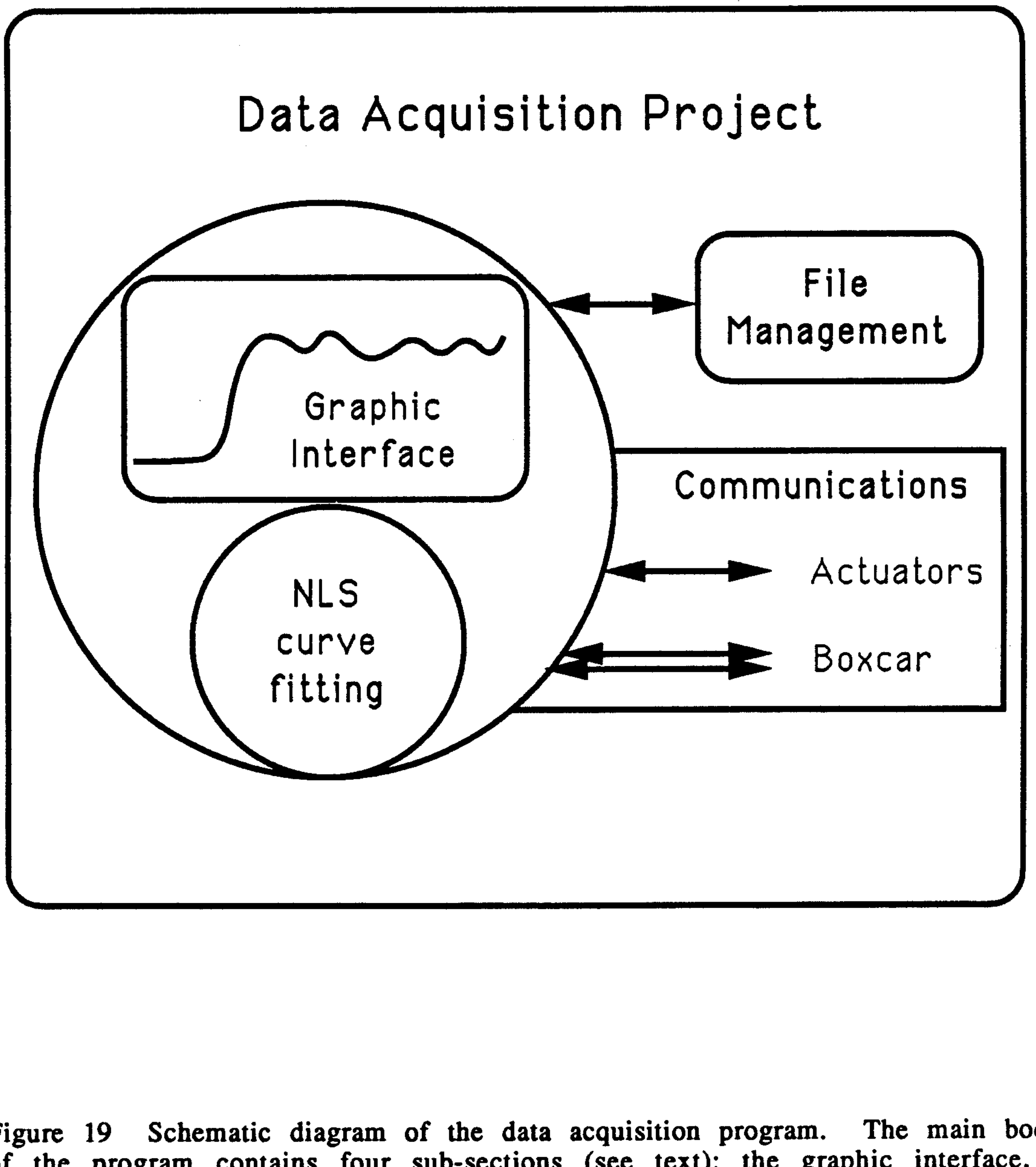
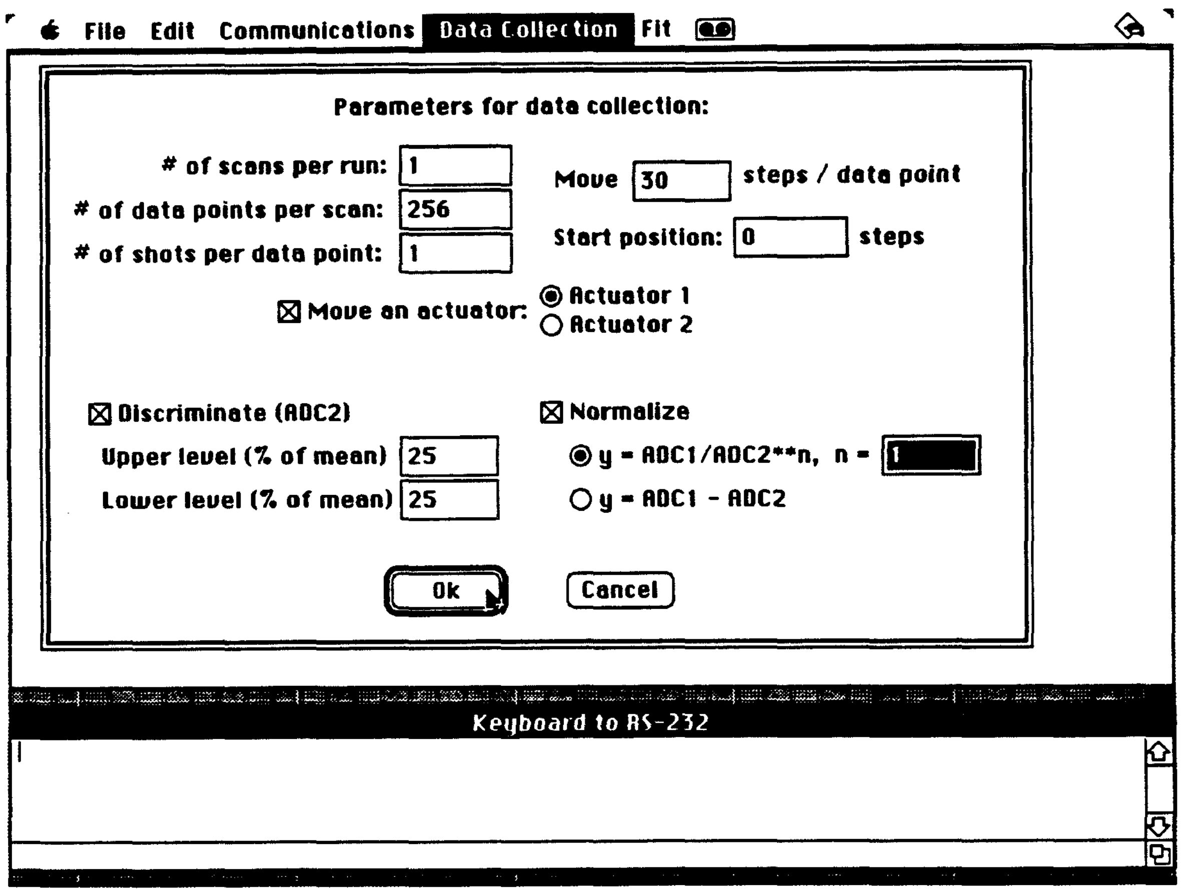


Figure 19 Schematic diagram of the data acquisition program. The main body of the program contains four sub-sections (see text): the graphic interface, the file management routines the communication software and non-linear least-square fitting routines.



Parameters for data collection:

# of scans per run: 1
# of data points per scan: 256
# of shots per data point: 1

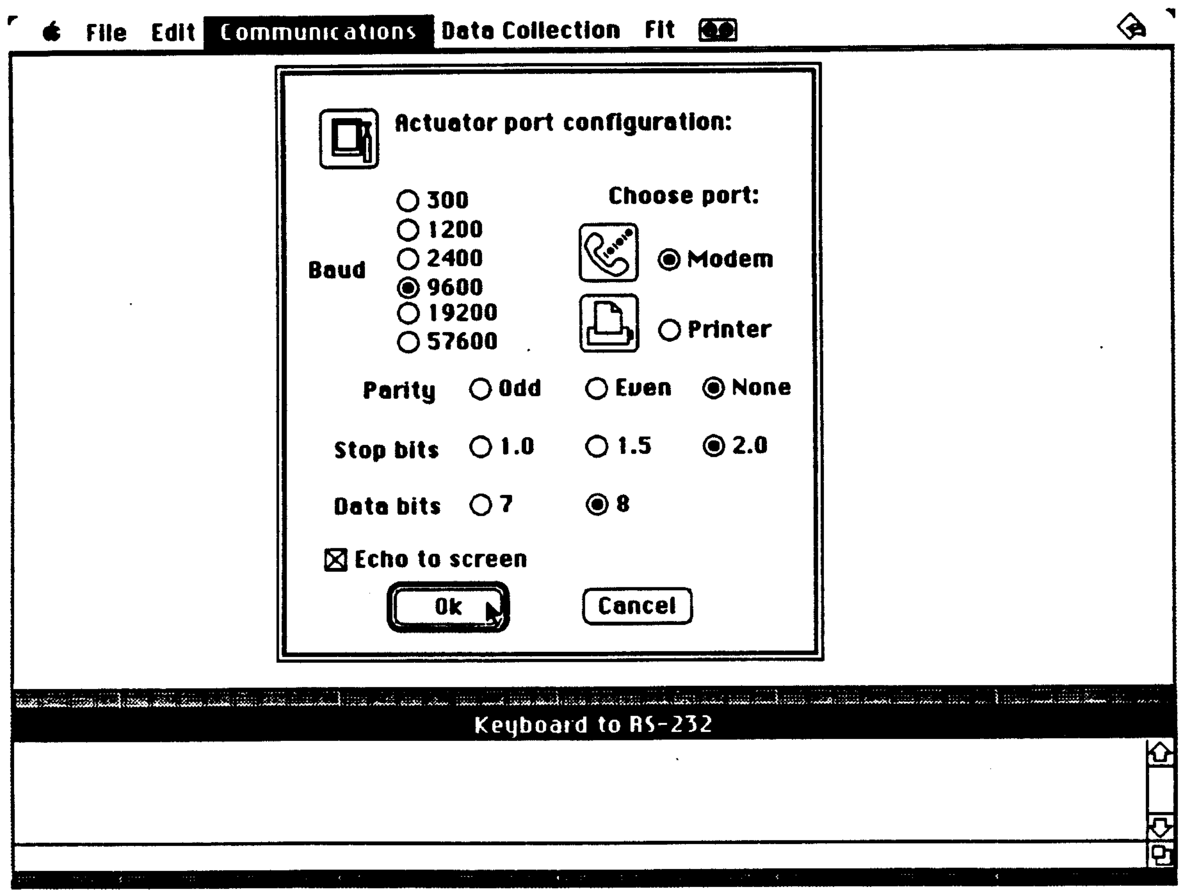
Move 30 steps / data point
Start position: 0 steps

☒ Move an actuator: ● Actuator 1 ○ Actuator 2

☒ Discriminate (ADC2)
Upper level (% of mean) 25
Lower level (% of mean) 25

☒ Normalize
● y = ADC1/ADC2**n, n = 1
○ y = ADC1 - ADC2

Figure 20 The data collection parameters as displayed by the program. The parameter dialog box, as shown, contains all the information relevant to a given data run.



Actuator port configuration:

Baud [300] [1200] [2400] [✓ 9600] [19200] [57600]

Choose port: [✓ Modem] [Printer]

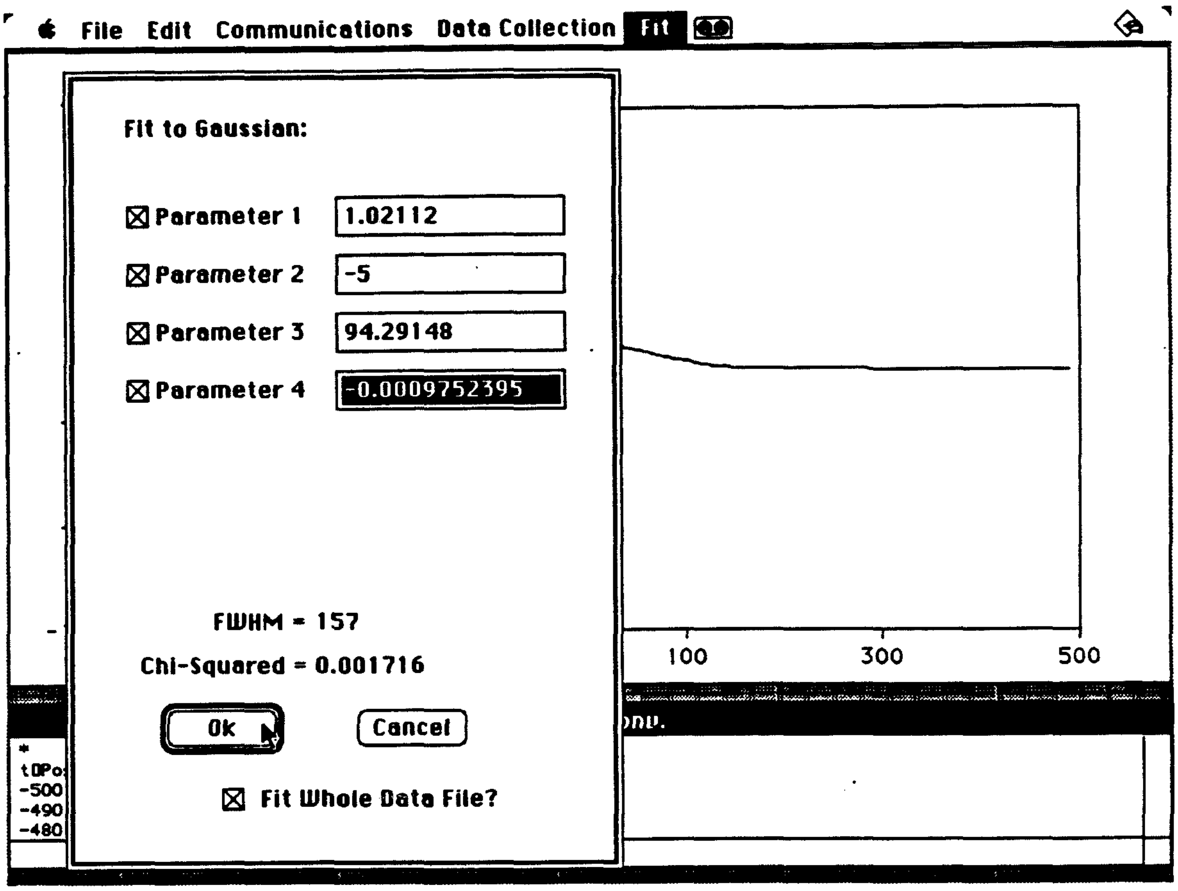
Parity [Odd] [Even] [✓ None]

Stop bits [1.0] [1.5] [✓ 2.0]

Data bits [7] [✓ 8]

[✓ Echo to screen]

Figure 21 The communication dialog box is part of the graphic interface that allows the user to select the parameters that manage the communications with the actuator controller.



Fit to Gaussian:

☒ Parameter 1 1.02112

☒ Parameter 2 -5

☒ Parameter 3 94.29148

☒ Parameter 4 -0.0009752395

FWHM = 157

Chi-Squared = 0.001716

☒ Fit Whole Data File?

Figure 22 The fit dialog box as displayed when fitting a Gaussian curve.

Chapter IV

Applications of FTS

The FTS technique, as described theoretically and experimentally here, has been successfully used in the study of different problems in molecular reaction dynamics. Real-time dynamics on repulsive, quasi-bound and bound potential energy surfaces have been recorded. FTS is now recognized as a valuable tool for the observation of transition-states in chemical reactions, for mapping bound and repulsive potentials through inversion of the temporal data, and for control of wave packet population. Of these applications only a few are included as representative examples.

The real-time observation of transition-states in the direct dissociation reaction of ICN, was the first demonstration of the FTS technique.[1] A full description of the FTS technique[2] including the first theoretical interpretations by R. Bersohn and A. H. Zewail,[3] as well as a full description of the ICN experiment[4] soon followed. For these first descriptions of the technique, each step was calibrated and scrutinized. In addition to the observation of transition-states of the reaction, a technique was developed to establish the zero-of-time of the reaction. This allowed us to clock the process of elementary bond breakage in real time on a direct dissociation reaction.[5] Soon after, R. B. Bernstein and A. H. Zewail developed a technique which allowed the direct inversion of the FTS data to construct the potential energy surface of the reaction.[6]

The best demonstration that the FTS technique was able to measure the events occurring during the dissociation, between reactants and products, came with the startling observations on the dissociation reaction of NaI by T. S. Rose, M. J. Rosker and A.H. Zewail.[7,8] The observations included the persistent wave packet oscillations caused by a Landau-Zener avoided crossings between the ionic and covalent states. These first observations were later further refined and published with full experimental and theoretical detail.[9]

The study of a system with more than one coordinate[10,11] gave the FTS technique more credibility as a universal tool and demonstrated the great flexibility afforded by controlling the time delay and wavelengths between the lasers. These studies allowed the observation of a dissociation reaction into two channels which could be distinguished by the amount of energy remaining on the non-reactive coordinate. These experiments have been further analyzed by semiclassical and quantum mechanical simulations by M. Gruebele and G. Roberts.[12] Further experiments on the ICN and HgI_2 systems making use of the polarization characteristics of the lasers allowed us to make observations of the nascent angular momentum and torque during the dissociation process.[13]

In addition to repulsive potential energy surfaces, bound potentials have also been investigated. The first experiments involved the B bound state potential of iodine.[14] These observations led to the realization that FTS techniques have, in some cases, advantages over conventional spectroscopy for the determination of a potential energy surface. The basis for the advantage of the FTS technique was that full rotational recurrences were observed completely separated in time (by two orders of magnitude) from the vibrational oscillations. These observations of molecular vibration and rotations were published in *Nature*.[15] The spectroscopic analysis of the data using quantum mechanics was published soon after by M. Gruebele et al.[16] A similar but simpler classical mechanical study was published by R. B. Bernstein and A. H. Zewail.[17]

Additional work on the alkali halides by P. J. Cong, A. Mokhtari and A. H. Zewail, led to the discovery of persistent wave packet oscillations during the dissociation process.[18] As for every interesting scientific problem, each of these projects has solved many questions but has open many more. A recent project on the interhalogens by M. H. M. Janssen, R. M. Bowman and A. H. Zewail[19] is one more of these examples. Presently new ideas have been born out of our experimental observations. We have used FTS techniques on liquid phase systems (unpublished), we have observed molecular motion and collisions at high pressures (unpublished), we have observed population control and the use of two and three photon transitions for the study of highly excited and one photon forbidden potential energy levels.[20]

All these experiments have been subject of many reviews,[21] but only a few representative examples are included in this thesis. Those included are: (1) applications to the dissociation reaction of ICN,[4] (2) applications to reactions with more than one coordinate,[11]

(3) application to bound potentials,[14] and (4) the role of alignment during dissociation reactions.[13]

References

1. M. Dantus, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 87, 2395 (1987).
2. M. J. Rosker, M. Dantus and A. H. Zewail, J. Chem. Phys. 89, 6113 (1988).
3. R. Bersohn and A. H. Zewail, Ber. Bunsenges. Phys. Chem. 92, 373, (1988).
4. M. Dantus, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 89, 6128 (1988).
5. M. J. Rosker, M. Dantus and A. H. Zewail, Science 241, 1200 (1988).
6. R. B. Bernstein and A. H. Zewail, J. Chem. Phys. 90, 829 (1989).
7. T. S. Rose, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 88, 6672 (1988).
8. M. J. Rosker, T. S. Rose and A. H. Zewail, Chem. Phys. Lett. 146, 175 (1988).
9. T. S. Rose, M. J. Rosker and A. H. Zewail, J. Chem. Phys. 91, 7415 (1989).
10. R. M. Bowman, M. Dantus and A. H. Zewail, Chem. Phys. Lett. 156, 131 (1989).
11. M. Dantus, R. M. Bowman, M. Gruebele and A. H. Zewail, J. Chem. Phys. 91, 7437 (1989).
12. M. Gruebele, G. Roberts and A. H. Zewail, to be published.
13. M. Dantus, R. M. Bowman, J. S. Baskin and A. H. Zewail, Chem. Phys. Lett. 159, 406 (1989).
14. R. M. Bowman, M. Dantus and A. H. Zewail, Chem. Phys. Lett. 161, 297 (1989).
15. M. Dantus, R. M. Bowman and A. H. Zewail, Nature 343, 737 (1990).
16. M. Gruebele, G. Roberts, M. Dantus, R. M. Bowman and A. H. Zewail, Chem. Phys. Lett. 166, 459 (1990).
17. R. B. Bernstein and A. H. Zewail, Chem. Phys. Lett., 170, 321 (1990).
18. P. J. Cong, A. Mokhtari and A. H. Zewail, Chem. Phys. Lett., 172, 109, (1990).

19. M. H. M. Jansen, R. M . Bowman and A. H. Zewail, Chem. Phys. Lett., 172, 99, (1990).
20. J. J. Gerdy, M. Dantus, R. M. Bowman and A. H. Zewail, Chem. Phys. Lett., 171, 1, (1990); R. M. Bowman, M. Dantus and A. H. Zewail, Chem. Phys. Lett. 174, 1 (1990).
21. A. H. Zewail and R. B. Bernstein, Chem. and Eng. News 11, 24 (1988); A. H. Zewail, Science 242, 1613 (1988); M. Gruebele and A. H. Zewail, Phys. Today 43, 24 (1990); M. Dantus and G. Roberts, Comments in Atomic and Molecular Physics, Accepted for publication (1990).

Chapter IV

Applications of FTS

1. Femtosecond real-time probing of reactions. II. The dissociation reaction of ICN

Femtosecond real-time probing of reactions. II. The dissociation reaction of ICN

Marcos Dantus, Mark J. Rosker, and Ahmed H. Zewail[a)]
Arthur Amos Noyes Laboratory of Chemical Physics, California Institute of Technology, Pasadena, California 91125

(Received 12 July 1988; accepted 11 August 1988)

Experimental results obtained for the dissociation reaction $ICN^* \to [I\cdots CN]^{\ddagger *} \to I + CN$ using femtosecond transition-state spectroscopy (FTS) are presented. The process of the I–CN bond breaking is clocked, and the transition states of the reaction are observed in real time. From the clocking experiments, a "dissociation" time of 205 ± 30 fs was measured and was related to the length scale of the potential. The transition states live for only ~ 50 fs or less, and from the observed transients we deduce some characteristics of the relevant potential energy surfaces (PES). These FTS experiments are discussed in relation to both classical and quantum mechanical models of the dynamical motion, including features of the femtosecond coherence and alignment of fragments during recoil. The observations are related to the radial and angular properties of the PES.

I. INTRODUCTION

Femtosecond transition-state spectroscopy (FTS) has been described in the first paper of this series (I).[1] Here, we present experimental results obtained with FTS for the dissociation reaction of ICN:

$$ICN^* \to [I\cdots CN]^{\ddagger *} \to I + CN\,. \tag{1}$$

In our first FTS studies of this reaction,[2] we reported the observation of the transition states $[I\cdots CN]^{\ddagger *}$ and gave the time scale for the formation of the free-CN fragment. In this paper, an account of the earlier work is given and a more complete description of the dynamics is presented using the simple classical and quantum mechanical theories discussed in I.

Studies of the dissociation reaction of ICN with FTS is attractive for several reasons. First, a wealth of information is known about the photodissociation of this linear (ground-state) molecule. Second, the fact that ICN is a triatomic molecule could allow us to more easily compare experiments with theory. Finally, the energy can be deposited in the I–CN bond, and the dissociation coordinate is that of a quasi-diatomic molecule. Nevertheless, there remains a number of unanswered questions about the dynamics of the ICN reaction, especially the shapes of the potential energy surfaces (PES) involved.

It has been established that excitation of ICN to the $\tilde{A}$ continuum[3] leads to dissociation via two channels[4]:

$$ICN + h\nu \to I(^2P_{3/2}) + CN(X\,^2\Sigma^+) \tag{2a}$$

$$\to I^*(^2P_{1/2}) + CN(X\,^2\Sigma^+)\,, \tag{2b}$$

producing iodine atoms either in the ground spin–orbit state $I(^2P_{3/2})$, or in the excited state $I^*(^2P_{1/2})$. For both channels (see Fig. 1), the CN fragment is produced in the ground electronic state; at least 99.9% of the CN fragment was found in its $(X\,^2\Sigma^+)$ state.[5]

In contrast to the cold electronic and vibrational excitation of the CN, the rotational excitation (e.g., for the 266 nm excitation, near the peak of the $\tilde{A}$ continuum) is rich and shows two distributions corresponding to the two channels. Photofragment angular distribution measurements,[6] product state distribution,[7] product yield of the I and I* channels,[8] sub-Doppler LIF measurements,[9] and spatial anisotropy and alignment experiments[10] have all been invoked to explain the dynamics of the different dissociation channels. On the basis of symmetry correlation diagrams it has been concluded[11] that the state which correlates with the I + CN channel is bent, while that which correlates with the I* + CN channel is linear. If the molecule is bent, because of the torque on the CN we expect a high degree of rotational excitation in the CN photofragment distribution. As shown recently,[9] for the I + CN channel, high rotational quantum numbers of the CN (N'') are selectively produced, whereas for the I* + CN channel the distribution peaks sharply at low N'' (low rotational energy).

An interesting question relating to the dynamics is the following: After absorption are these two channels independently produced? Or is the absorption to one state (linear), which by curve crossing leads to two channels? Nadler *et al.*[9] suggested that only one state is involved in absorption, and that this absorption is to a linear state having predominantly parallel character. This linear state(s), which correlates with I* + CN, crosses a bent state that produces I + CN. Zare's group,[10] by measuring the alignment of the photofragment, concluded that the dynamics of the curve crossing between linear and bent surfaces takes place in the Franck–Condon region. Further support comes from the calculations by Goldfield *et al.*[12] of the rotational distribution and the yield of I and I*, using nonadiabatic interactions between the ICN surfaces.[13]

Our first subpicosecond[14] and FTS[2] experiments on ICN, as well as all experiments described here (except for some at 285 nm), were made at 306 nm. At or near this wavelength, which is much to the red of the absorption peak and corresponds to a total available energy of 6000–7000

[a)] John Simon Guggenheim Foundation Fellow.

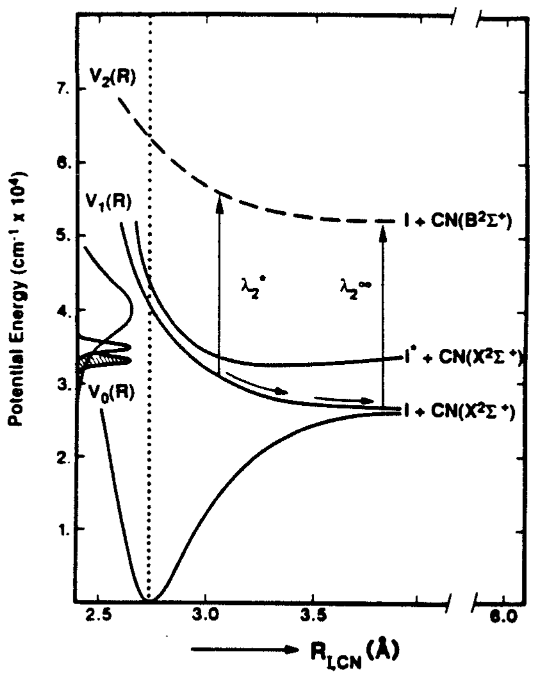


FIG. 1. Schematic of FTS and the potential energy surfaces of ICN. The pump pulse causes a vertical transition from V_0 to V_1, and the subsequent motion on this surface is indicated. The transition state is probed by λ_2^*, or the final state by λ_2^∞, which are shown by the respective arrows from V_1 to V_2. The absorption spectrum of ICN and the pump energies used are sketched on the left. Note that the difference between λ_2^* and λ_2^∞ is not to scale, and that the van der Waals wells are not shown.

cm^{-1} above dissociation, only the I + CN channel is produced,[15] which simplifies the problem. (The spin–orbit splitting of iodine is 7600 cm^{-1}.) For these wavelengths, the rotational distributions are known, and they peak around $N'' \sim 25$.[15] Simple kinematics can actually account[16] for these rotational distributions at the energies of interest for this "direct" (i.e., < 1 ps) dissociation.

In the past, the source[17] of information on the (average) lifetime $\langle\tau\rangle$ of dissociation came from measurements of angular distribution(s) and from the calculated average rotational period $\langle\omega\rangle$ of the parent molecule, as illustrated for the case of ICN by Ling and Wilson.[6] Using the spatial anisotropy parameter β[17,18] one may estimate $\langle\tau\rangle$. For example, for a parallel-type transition and for recoil along the internuclear axis:

$$\beta = 2\,\frac{1 + (\langle\omega\rangle\langle\tau\rangle)^2}{1 + 4(\langle\omega\rangle\langle\tau\rangle)^2}\,. \tag{3}$$

Knowing β and computing $\langle\omega\rangle$ ($\sim 1 \times 10^{12}$ rad s^{-1} for ICN at 300 K),[9] one deduces $\langle\tau\rangle$. This clever idea uses the rotation of the molecule as an average clock; if the molecule dissociates before it rotates then the anisotropy is very high. For ICN, $\langle\tau\rangle$ is estimated to be a few hundred femtoseconds.

There are several points, however, that must be considered. First, coupling between surfaces and/or changes in recoil direction make interpretations less straightforward. There is also the symmetry factor[19] in β which must be known. In applying Eq. (3) to ICN, we note the following: The rotational period $\langle\omega\rangle$ should depend on the temperature and, accordingly, β should generally change with temperature if $\langle\tau\rangle$ is the same. For ICN, the angular distribution was found to be the same for room temperature (300 K) photodissociation as it was for supersonically cooled (~ 10 K) ICN.[20] The period $\langle\omega\rangle$ changes in these two experiments by a factor of 5, since the most probable $J_{\rm ICN}$ are 31 and 6, respectively, at the two temperatures ($J\hbar = I\omega$). Either $\langle\tau\rangle$ does change or the dynamics of the dissociation are independent of the parent rotation. However, in the former instance if $\langle\omega\rangle\langle\tau\rangle \ll 1$, as is the case here, then Eq. (3) is not very sensitive to changes in either $\langle\omega\rangle$ or $\langle\tau\rangle$. Another point is that slight differences in the magnitude of β have been attributed in the literature to changes in the direction of the velocity vector in the recoil process. As discussed by Zare,[10] in the case of ICN this bending during recoil may change the magnitude of the CN rotational angular momentum but will not change its direction (N angular momentum remains opposite to L, orbital angular momentum, and thus perpendicular to the ICN plane). Accordingly, while β may decrease, the angular momentum alignment should remain constant, which is in contradiction with the alignment experiments of O'Halloran *et al.*[10] Therefore, it is of interest to follow the real-time femtosecond dynamics of this dissociation reaction by observing the transition states and the development of the free fragments.

Since the recoil velocity for the reaction at energies of interest is typically ~ 0.025 Å fs^{-1}, the time resolution of these FTS experiments allows us to view, with good resolution, a few angstroms of recoil. Also, on this time scale we can study the loss of alignment and the development of the CN angular momenta as a function of time. Three types of experiments are reported here: (i) clocking of the process of I–CN bond breaking and direct determination of τ for different recoil velocities; (ii) characterization of femtosecond transients for the transition states $[\mathrm{I\cdots CN}]^{\ddagger *}$; and (iii) studies of recoil fragment coherence and angular momenta using polarized femtosecond pulse arrangements. The results are discussed in relation to the radial and angular parts of the PES and to the real-time dynamics that produce the I and CN fragments.

The organization of the paper is as follows: In Sec. II, we describe the experimental details specific to the FTS of ICN. This includes a discussion of a number of characterization and diagnostic experiments that we have performed. In Sec. III, we give the results and discuss the different types of experiments performed in relation to clocking, transition-state characterization, femtosecond coherence and alignment, and the description of the PES. Our conclusions are summarized in Sec. IV.

II. EXPERIMENTAL

The experimental methodology of FTS was presented in paper I of this series. In this section, we provide specific details pertinent to the FTS experiments of the ICN reaction.

A. FTS experiments

As discussed in paper I, the femtosecond pulses were generated from a colliding pulse mode-locked ring dye laser (CPM), and were amplified by a Nd:YAG-pumped pulsed dye amplifier (PDA). The output pulses of this combination had $\sim 300\,\mu$J energy and 20 Hz repetition rate.

Conversion of this light to the pump frequency was accomplished using either of two procedures. For most of our ICN experiments, the pump wavelength was chosen to be at ~ 306 nm, which could be conveniently produced via second harmonic generation of a portion of the PDA output. This was performed by focusing the light into a thin (500 μm), phase-matched KD*P crystal. Because of the high peak intensity of the femtosecond pulses, the conversion efficiency into the second harmonic exceeded 10%. The second pump generation procedure yielded femtosecond light centered near 285 nm. This was accomplished by sum frequency mixing in a KD*P crystal of a portion of the PDA output with residual, synchronous 532 nm radiation from the YAG pump laser. The efficiency of this conversion was much lower, being limited by the onset of optical damage to the crystal by the nanosecond YAG pulses.

The probe pulses measured the absorption of the system at or off-resonance to the free-CN transitions near the bandhead at 388.5 nm. In our experiments, the probe was generated by mixing together in KD*P the remainder of the PDA output with 1.06 μm light from the YAG laser (see Fig. 2). The probe pulses were tunable over a limited range of wavelengths (387 to 396 nm) by careful adjustment of the cavity elements of the CPM, specifically the positioning of the intracavity prisms and of the saturable absorber jet. (Adjustment of the phase-matching angle for the nonlinear mixing crystal was also required.)

The spectral resolution ($\Delta\nu$) of the probe could be adjusted in order to observe the transition states of ICN. Because of the inverse relationship between spectral and temporal pulse width, however, as $\Delta\nu$ was made smaller, the temporal resolution (Δt) become poorer. Care was exercised to insure that the pulses were chirp free (transform limited), so that $\Delta\nu\,\Delta t = 0.3$ to 0.5. However, the effect of nontransform limited pulses was also investigated.

The pump and probe beams, with proper attenuation and parallel or perpendicular polarization, were delayed in time relative to one another in a Michelson interferometer, and were collinearly recombined and focused into the reaction chamber. For these reported experiments, the chamber was a custom-made cell arrangement (see I). The ICN was kept at room temperature and the pressure was maintained at 0.1–0.3 Torr. The materials used in the cell construction [Teflon, Delrin, stainless steel (304), and quartz] were tested for inertness. Laser-induced fluorescence (LIF) of the CN fragment was detected, and was collected at right angle to the direction of the pump/probe pulses through a $f/1.5$ lens arrangement. To suppress scattered light, black Teflon baffling was invoked, and the entrance and exit optical windows were oriented at Brewster's angle. Spurious light was also rejected by dispersing the LIF through a 0.34 m monochromator. A Hamamatsu R1527 photomultiplier tube (PMT) detected the LIF. A digital boxcar integrator averaged that portion of the PMT voltage which occurred within an adjustable temporal gate. The entire experiment was controlled by a microprocessor, which adjusted the optical delay line of the Michelson interferometer and acquired and numerically averaged together the boxcar output.

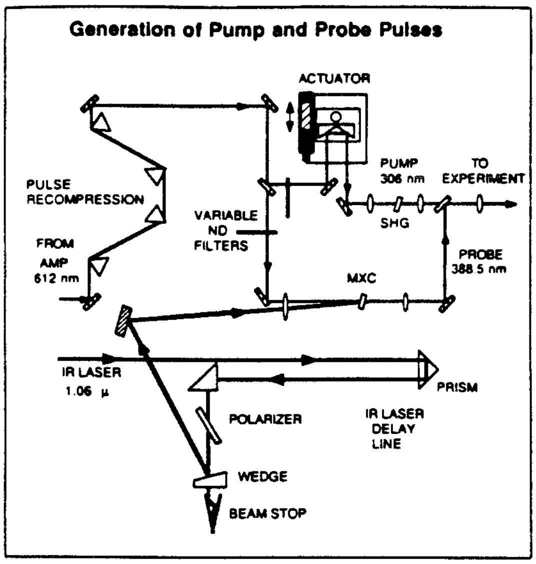


FIG. 2. Schematic of the pump/probe generation scheme. The output of the amplifier is first recompressed and then split into two equal beams. Each arm is then converted to the appropriate wavelength, by second harmonic generation (SHG) for the pump, and by sum frequency mixing (MXC) for the probe. The beams are then colinearly recombined. The delay time of the Michelson interferometer is provided by a precision actuator. (The ~ 285 nm pump generation technique, not shown, is similar to that of the probe generation.)

Characterization of the pump and probe pulses was considered crucial, and so for each data set, several additional measurements were made. First, the spectrum of the pump and the probe were each taken. Also, the cross correlation of the pump with the probe was measured, using the technique of difference frequency generation (DFG) in a nonlinear crystal ($\omega = \omega_{\mathrm{pump}} - \omega_{\mathrm{probe}}$). The DFG light near 1.45 μm was detected with a germanium photodiode. Further, we determined the overall temporal response function of our apparatus and the zero-of-time (defined by the temporal midpoint of the pump pulse).

Determination of the response function and the exact zero-of-time was critical to many of the experiments, and was accomplished as follows: The reaction chamber was constructed with an ionization electrode, which could be translated into the reaction region without movement of the cell or loss of vacuum. The cell also contained a second sample holder, containing N,N-diethylaniline (DEA). The response function measurement was made by evacuating the chamber of ICN and replacing it with DEA (at 0.3 Torr or less), positioning the ionization electrode in the reaction region, applying a dc voltage to the electrode, and measuring the resulting multiphoton ionization (MPI) current. Ionization of the DEA required both a pump and a probe photon, so that the MPI signal as a function of the time delay between the pump and probe gave the detection response function.[14] We have also verified this by comparison of the DEA experiment results with the numerically integrated results of the cross correlation of the pump and probe. As discussed in paper I, the $t = 0$ point was the time at which the MPI signal reached one-half of its limiting value. The response function measurement was made completely *in situ*, with no repositioning of the cell or adjustment of any optical elements.[21]

DEA was chosen for the calibration of the $t = 0$ because the intermediate lifetime (aniline type) of the state reached by the pump pulse is long lived. Hence the signal will rise "instantaneously" with our pulse and will reach a plateau at long times, as observed experimentally. By instantaneous, we mean that it is as fast as any electronic transition response. We have considered, however, the possible existence of initial dephasing or redistribution, which would cause some delay in the MPI signal. However, in our experiments the DEA is excited very near the leading edge of absorption ($\lambda_{max} = 303$nm)[22] so that IVR is on a very long time scale.[23] Furthermore, rotational dephasing for this molecule at room temperature using the approach of Ref. 24 is ≈ 1 ps. All of these times (nanosecond to picosecond) are much longer than the time scale of our experiment. The *precision* of the $t = 0$ calibration is fixed primarily by the experimental signal-to-noise ratio, but ultimately, the *accuracy* of the calibration will be limited by the response time of a molecule like DEA. Finally, it should be noted that since the cross section is $\sigma = 1 \times 10^{-18}$ cm^2 for DEA,[22(b)] and the flux is $\Phi \sim 5 \times 10^{14}$ photons pulse^{-1} cm^{-2}, then less than one in a thousand molecules is in the excited state. Hence, there is no saturation effect.

B. Diagnostic experiments

Numerous diagnostic experiments were performed in order to insure the proper treatment of the data. First, to establish the linearity of the signal with respect to the intensity of both the pump and the probe beams, the dependence of the LIF signal on the pump or probe energy was recorded. This measurement was performed under several experimental circumstances, including: (1) with an on-resonant probe ($\lambda_2 = \lambda_2^\infty$) and long time delay ($t > 1$ ps); (2) with an off-resonant probe ($\lambda_2 = \lambda_2^*$) and long time delay; and (3) with an off-resonant probe and with the time delay set to maximize the FTS signal ($t = t^*$). These results are plotted on a log–log scale in Fig. 3. For each case, the data falls on a line with a slope near unity, which is evidence for the linearity of the FTS experiment (indicating that the ICN is photolyzed by a single pump photon, and that the fluorescence from the CN is induced by a single probe photon) as well as the detection apparatus. As a further test, FTS transients were acquired both on- and off-resonance over a wide range of pump and probe powers (factor of ~10). We have found no evidence of saturation in any of these measurements. These results are consistent with the sensitivity calculation presented in Sec. V A of paper I. Estimation of the total flux of photons in the measured LIF signal gives 10^4–10^5 photons per pulse, which is consistent with these calculations (after taking into account various factors such as cross sections of the transitions, detector sensitivity, yields, etc.).

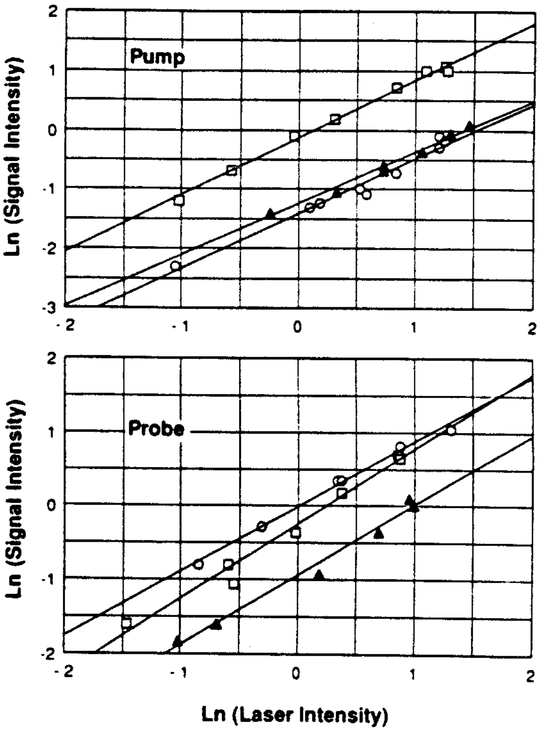


FIG. 3. Power dependence of the LIF signal. The observed LIF is plotted as a function of the pump or probe intensity on a log-log scale. Key: open squares: on-resonance transients; solid triangles and open circles: off-resonance transients for different time delays. The lines shown are best fits to the data. Note that, in all cases, the slopes of the lines are near unity.

In order to characterize the fluorescence lifetime of the LIF signal of the CN, we scanned the boxcar gate position and recorded the nanosecond decay of the CN radical. The recorded transient fit well to a single exponential decay, and the time constant was of the same order as previously reported values for the CN emission lifetime.[25]

Transients were also obtained for ICN pressures in the range 0.05–0.5 Torr to check for any collisional effects. The only trend seen in this data was the expected decrease in the overall signal level as the pressure was reduced. The average collision time calculated at 0.2 Torr and room temperature is ~0.1 μs. The effect of collisions on the dynamics of the reaction (especially the femtosecond dynamics) can be ignored.

The pump and the probe each contributed a background level to the observed data, which was independent of delay and superimposed on the delay-dependent FTS transient. A part of this "dc" level was determined to be caused by weak multiphoton absorption of the ICN by the pump light. This was easily reduced to < 10% of the peak signal by attenuation of the pump. The remaining contribution (also ~ 10% of the peak level) was associated with scattered probe light. As already described, the techniques to discriminate against probe scatter included the use of baffles, Brewster windows, and temporal gating. Typical enhancements observed in these experiments when both the pump and the probe overlapped temporally were 10:1.

Since the scattered light persisted for only the detector response time (< 5 ns), the gating of the boxcar was typically adjusted to open afterwards, and remain open for the rest of the fluorescence lifetime. The effect of inclusion in the observed FTS transients of the scattered probe light was studied by collecting transients with the following gate arrangements: (1) with the gate overlapping the early portion of the fluorescence (including the scattered probe light, if it existed); (2) with the gate opened only at the very end of the fluorescence; and (3) with an extremely wide gate, beginning well before the pump and probe pulses and extending for several fluorescence lifetimes. In each case, the shape of the FTS transient obtained was basically unchanged. As a further test, the scattered light level was increased by an order of magnitude by placing a card at the output Brewster window. Again, the measured FTS transient was unaffected (aside from the increase in the background level).

To check for polarization effects, we have verified that similar (but not identical) FTS transients were obtained for both parallel and perpendicularly polarized pump and probe beams. In all cases the probe beam polarization was oriented along the direction of the LIF collection,[26] and the pump beam polarization was adjusted with the rotation of a zero-order, half-wave retardation plate. The differences observed between these data sets will be discussed below.

The possibility of coherent interaction [see, e.g., Refs. 27(a)–27(c)] between the pump and the probe was considered, and was ruled out. First, the transient shape is very sensitive to the probe wavelength within a range of a few nanometers; on-resonance (λ_2^∞) we observe only a delayed rise, and off-resonance (λ_2^*) we see systematically a build-up and decay with a shift in delay time depending on λ_2^*. Second, the pump and probe are much different in wavelength, by ~ 80 nm. The very large difference in the respective wavelengths gives a coherence length for such an energy separation of only 1.4 μm, which is much smaller than the length of the interaction region (the confocal parameter of the optical pulses = 15 mm). Equivalently, the coherence time is 5 fs, which is much shorter than the observation times. Because of the observed linearity of the signal (weak signal limit), the large difference between pump and probe wavelengths and the relatively small cross section ($\sim 4\times 10^{-20}$ cm^2) at ~ 306 nm, we can also exclude the Stark shift produced by our pulses. [See, e.g., Ref. 27(d).]

Finally, the cyanogen iodide used in this study was prepared by vacuum sublimation of ICN (Aldrich, 95%), which was repeated until the resulting material was completely colorless. The composition and purity of the compound was tested by mass spectroscopic methods. The *N,N*-diethylaniline (Aldrich, 98%) was fraction distilled until it became colorless and then was extensively outgassed.

III. RESULTS AND DISCUSSION

A. Clocking experiment

In the clocking experiments, described in paper I, the wavelength of the probe is tuned to be on-resonant with the final product state. In this case, $\lambda_2 = \lambda_2^\infty = 388.5$ nm, which is the transition wavelength of the bandhead for the *P* branch of free-CN. Each data set collected consisted of an ICN transient, a DEA multiphoton ionization transient, and a probe spectrum.

Typical clocking results are presented in Fig. 4. The hollow circles in the figure are the experimental response function for the system, as determined by the DEA measurement. The zero-of-time is found from this data by the requirement that one-half of the limiting value is achieved at $t = 0$. The solid squares in the figure show the FTS transient obtained with a 306 nm pump pulse. The femtosecond "time-of-flight" of the fragments is $\tau_{1/2} = 205 \pm 30$ fs. (The uncertainty here is an estimate from the scatter for several measurements of the observed $\tau_{1/2}$ values.) This value was obtained by least-squares fitting of the curves to the same function (solid lines). But independent of the fitting procedure, the delay of the free fragment is evident and measurable.

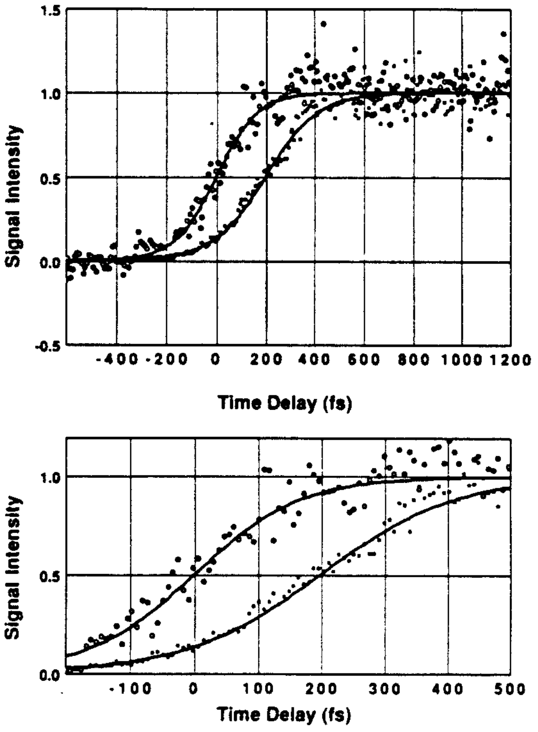


FIG. 4. Typical clocking experiment results with transform-limited pulses. The ICN transient (solid squares) in this case shows an observed delay of $\tau_{1/2} = 205 \pm 30$ fs from the DEA transient (open circles). The lines are fits to the data. The figure at the bottom shows an expanded scale near $\tau = 0$.

Measurement of $\tau_{1/2}$ gives the time required to break the bond between the I and CN fragments. As discussed in paper I, the results of the clocking experiment are relatable to the shape of the PES, since $\tau_{1/2}$ is the length of time required to move with a recoil velocity $v(t)$ from the initially excited configuration at R_0 to the optically coupled region (OCR) of the probe. The terminal recoil velocity can be obtained from the translational energy in the center-of-mass frame: $E = 1/2\,\mu\,v^2$.

As a first approximation, suppose that the fragments moved on this path at the terminal recoil velocity, $v = 0.026$ Å fs^{-1}.[28] At this velocity, the increased separation betweeen the fragments in $\tau_{1/2}$, from their initial separation, would be 5.3 Å. However, this value is essentially an upper bound, since the velocity of recoil begins at zero and changes with distance.

A more useful approach is to assume some functional form, e.g., an exponential repulsion between I and CN, and to consider the reaction as a classical motion on this potential. This problem was considered in paper I, and it was shown, under the approximations made, that for the clocking experiment ($t^* = \infty$):

$$\tau_{1/2} = (L_1/v)\ln(4E/\gamma)\,, \tag{4}$$

where γ is the half-width of the energy distribution of the probe pulse, and E is the energy available above dissociation. Knowing v, E, and γ, we calculate from the measured $\tau_{1/2}$ that $L_1 = 0.8$ Å. This length parameter is obtained assuming the one-dimensional potential, with no centrifugal term to account for the orbital angular momentum.

This brings to focus the problem of how to define the dissociation time. If it were defined as the time for the pair of fragments to reach a separation at which their interaction energy is negligible, the dissociation time would depend on the sensitivity with which that energy could be measured; an infinite time is required to reach infinite separation. A more invariant definition can be obtained through the following approach: Consider the difference between the time taken for the fragments to separate from R_0 to R as they travel along the actual potential, minus the time which would have been required had the fragments traveled at the terminal velocity throughout. Then we may define the dissociation time τ_d as the limit of this difference as $R \to \infty$. On a repulsive surface, this leads to $\tau_d = (L_1/v)\ln 4$.[29] While this τ_d is well defined, it is obtainable only from a hypothetical experiment requiring the measurement of minute differences in long arrival times. See paper I, Fig. 14.

Equation (4) defines the physics of the problem more clearly: dissociation occurs when the fragment is spectroscopically identifiable as free (or nearly free) from the other fragment. Hence, the effect of E, γ, and v is understood (see I). Tests of these dependences will be discussed below. The value of $L_1 = 0.8$ Å obtained from Eq. (4), it should be noted, represents the length at which the potential falls to $1/e$ of its initial value; the distance at which the bond is "completely" broken is several times this value. As discussed in paper I, $\tau_{1/2}$ is the time for the potential to drop to the value of γ (in this case, 40 cm^{-1}). Accordingly, the bond is broken at 4 Å. As expected, this value for the length parameter is smaller than the previously discussed, unrealistic value of 5.3 Å, which assumed the terminal velocity throughout the fragment motion.

By increasing the pump energy, and so reducing the initial internuclear separation R_0, we can test the effect of recoil velocity and available energy. In this case, the pump wavelength was reduced from 306 to 285 nm. Two competing effects are involved: On the one hand, the distance from R_0 to the probe OCR becomes greater, which would tend to increase $\tau_{1/2}$. On the other hand, as we increase E, the recoil velocity increases and $\tau_{1/2}$ will shorten. Since the rotational internal energy of the fragments is known (~ 2000 cm^{-1} at 285 nm vs ~ 1200 cm^{-1} at 308 nm),[12] the available (and translational) energy is increased, so that the final recoil velocity is larger. For ICN, at 285 nm, the I* channel is energetically possible, and there exists the possibility of an avoided crossing between the two channels that lead to I and I*. Such a crossing, if it occurred, might substantially affect the dynamics of the reaction, leading, for example, to long delays or to oscillations (resonances) like those observed for the alkali halides.[30]

Our results indicate that tuning the pump to 285 nm is associated with a net reduction in $\tau_{1/2}$, to 160 ± 30 fs. This suggests that the predominant effect involved is due to the increased velocity of fragment recoil. Considering Eq. (4), $\tau_{1/2}$ at the two available energies are consistent with the same length parameter of the potential. No direct evidence has yet been obtained that the crossing from the I* channel to the I channel delays the appearance of free-CN or leads to resonances. This is consistent with the idea that such crossings are at shorter R_0, near the Franck–Condon region. However, work is continuing in this area, and the possibility exists that these phenomenon may be observed if the pump wavelength is tuned to wavelengths shorter than 285 nm or if the pulse widths are further shortened.

The effect of the spectral width of the pulses (γ) was studied in two ways: (1) by changing $\Delta\nu$ and Δt of the pulses over a limited range in a way that maintained transform-limited pulses (accomplished by adjustment of the CPM laser), and (2) by spectrally broadening the pulses (by slight saturation in the PDA), which produced pulses with $\Delta\nu\,\Delta t \gg 1$. Even though the effect of γ in Eq. (4) is relatively insensitive, we observed a consistent trend for $\tau_{1/2}$ vs γ for the many experiments we did. These results were also consistent with previous measurements[14] made with subpicosecond pulses, where the full rise was found to be in ~ 600 fs. We have noted, however, that the existence of a chirp in the optical pulses could affect the timing of an on-resonance rise, because the exact $t = 0$ may not be well defined when each wavelength component of the pump arrives at a slightly different time. This would not seem to be the predominant effect in our nontransform limited data, since we have repeated the experiments for probe spectra centered at slightly different wavelengths and these obtained values of $\tau_{1/2}$ were self-consistent.

It is interesting to note the similarity in shape of the two traces in Fig. 4. As detailed in paper I, the FTS transient should approximately equal the response function (which is the integral of the cross correlation) in the limit where the intrinsic spreading of the wave packet on V_1 is ignored or, more exactly, if the spreading is on a shorter time scale than the duration of the pump and probe pulses. Within the experimental errors, the two traces are not significantly different, so that at this time we cannot address with confidence the question of packet dispersion.

In conclusion, these clocking experiments provide an operational way to obtain a characteristic femtosecond time-of-flight $\tau_{1/2}$ for a chemical bond. This $\tau_{1/2}$ can be related to the dynamics of bond breaking and the "window" (the OCR) opened on the potential by the probe.

B. Detuning experiments: To the red

For ICN, tuning the probe wavelength to the red of the bandhead at $\lambda_2^\infty = 388.5$ nm leads to the observation of femtosecond-lived transition states.[2] In Fig. 5, the results for FTS experiments for several values of $\lambda_2 = \lambda_2^*$ [i.e., $A(t; \lambda_{2(a)}^*, \lambda_{2(b)}^*, ..., \lambda_2^\infty)$] are presented. The on-resonant case, $\lambda_2 = \lambda_2^\infty$, was discussed in the last section. The FTS transient behavior characteristically was a simple rise (after a delay $\tau_{1/2}$) to a limiting value, which represented the build-up of population into the final state. The other transients in Fig. 5 shows some representative results obtained for off-resonant probe wavelengths, $\lambda_2 = \lambda_2^*$. In the detuned experiments, the behavior is seen to be a rise followed by a decay to some asymptotic level. The height of the asymptote is a function of the probe wavelength. As predicted in paper I, this level decreases as λ_2 is tuned away from λ_2^∞. Note that for $\lambda_2^* = 391.4$ nm in Fig. 5, the decay of the transient is virtually complete. Such behavior has been repeatedly observed in many experiments not shown here (we only present these four typical sets), and is consistent with the measured probe spectrum as discussed below.

The decay in some cases is on a longer time scale than the pulses, but in other cases is within our time resolution, depending on λ_2^*. We use the experimentally measured cross correlation to correct for this convolution effect. To illustrate the effect of pulse convolution on our transients, we present in Fig. 6 some simulations.

We now apply the simple theories discussed in paper I to these detuned experiments. The kinetic model in Sec. V A of I describes the height of the asymptote by the inclusion of a tuning parameter β. Tuning of the probe as in Fig. 5 would correspond to changing β from 0 (completely on-resonant case) to 1 (completely off-resonant case). The data presented in Fig. 5 can be well fit by the six-level approach (see Fig. 11 of paper I). The spontaneous lifetimes obtained from the parent to the transition state and from there to the final product state ($1/k_{bc}$ and $1/k_{ce}$, respectively), are of the expected order-of-magnitude, $\sim 10^{-13}$ s. As pointed out in I, the use of the kinetic approach is heuristic because the continuum of transition states is replaced by a single, discrete level.

A more meaningful description is obtained using the

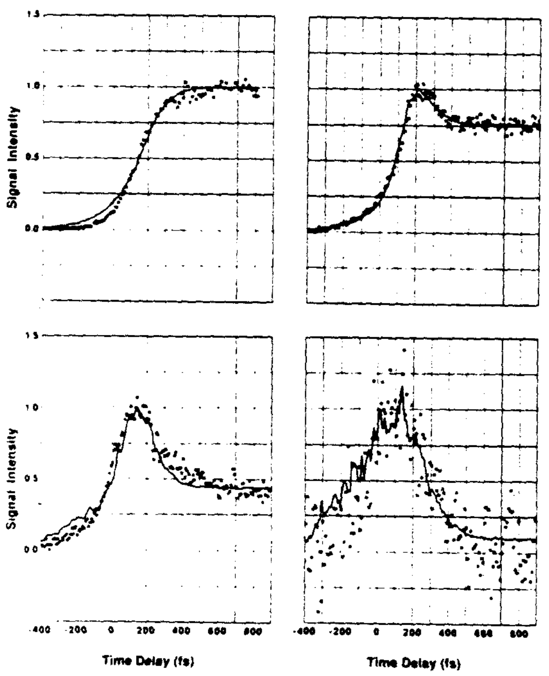


FIG. 5. Typical ICN transients as a function of probe wavelength. The probe wavelengths were (a) $\lambda_2^* = 388.9$ nm, (b) $\lambda_2^* = 389.8$ nm, (c) $\lambda_2^* = 390.4$ nm, and (d) $\lambda_2^* = 391.4$ nm. The solid lines were generated by fitting the classical model of Eq. (5) convolved with the measured experimental response for each transient.

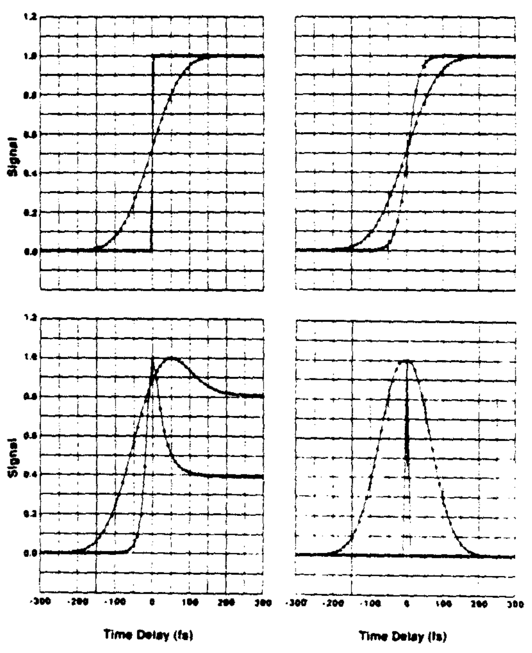


FIG. 6. Simulation of the effect of convolution on the transients. The numerical results (solid circles) are the convolutions of a Gaussian pulse of 150 fs width with the following assumed shapes (open squares): (a) an ideal step function; (b) Eq. (5) of the text, with $t^* = \infty$ (on-resonance); (c) Eq. (5) of the text, with $t^* < \infty$ (off-resonance); and (d) a delta function. Note that $\tau_{1/2}$ is not affected by the convolution in cases (a) and (b).

classical mechanical model[29] presented in I. Here, we shall invoke two assumptions which will be discussed below: the excited state PES (V_1) has an exponential shape, and the upper state potential satisfies a condition to be discussed shortly. The absorption of the probe photon as a function of time is then given by Eq. (20) of paper I:

$$A(t) = \frac{C}{\gamma^2 + W^2(t,t^*)}, \tag{5a}$$

where

$$W(t,t^*) = E\left\{\operatorname{sech}^2\left[\frac{vt}{2L_1}\right] - \operatorname{sech}^2\left[\frac{vt^*}{2L_1}\right]\right\}. \tag{5b}$$

In this expression, t^* is defined to be the time at which the absorption achieves a maximum, C is a constant, and L_1 is the distance scale of V_1.

The solid lines in Fig. 5 are the best fit results obtained by the convolution of Eq. (5) with the experimentally measured cross correlations. The underlying (deconvolved) absorptions are shown for each of these fits in Fig. 7. The only parameters allowed to vary in these fits are t^* and L_1 (γ is determined by the probe spectrum), and the results are presented in the caption.

These fits raise a number of interesting issues. First, there is good consistency among all the best-fit values for L_1 ($\simeq 0.8$ Å) in providing t^* values that are systematically expected to occur at shorter times. In addition, this value is in agreement with that obtained for L_1 using the clocking approach, $t^* \to \infty$ (see last section). The closeness of these fits to the measured data implies that

$$V_2^0 \exp\{-[R(\tau_{1/2}) - R_0]/L_2\} \ll V_1^0 \exp\{-[R(\tau_{1/2}) - R_0]/L_1\}.$$

This condition is satisfied either by making L_2 less than L_1, or else by $V_2^0 \ll V_1^0$, and is consistent with our observation of a red shift. We have made simulations of the more general case to test the effect of V_2^0 and L_2 and found consistency in the derived value of L_1. (We will detail this study later.) The potential surfaces of ICN cannot be expected to be exactly exponential in shape throughout, because of the existence of long-range interactions and van der Waals wells. However, the exponential shape discussed gives an idea of the distance scale involved. The value of L_1 obtained here is about a factor of 4 larger than the value (~ 0.2 Å) obtained by several authors[31] from the absorption spectrum continuum in the Franck–Condon region. This is very reasonable in light of the fact that, in the Franck–Condon region (at short R's), the potential is very steep, and in the long-range part we expect a different slope (less steep). The inner shell repulsions usually have a much shorter range than the attractive forces, which might cause the net repulsion to change its shape at the longer distances.

We now consider the variation with respect to t^* in this data. Note that in Figs. 5 and 7, the time delay is given in absolute units. Determination of the $t = 0$ timing was made for each data set using the multiphoton ionization technique previously described. The time delay at which the FTS signal achieves its maximum moves towards earlier times as the detuning is increased. The systematic variation in the peak position is even more apparent in the deconvolved results, Fig. 7. This is a crucial observation, required by the models of FTS discussed in paper I. Because the transition induced by the probe from V_1 to V_2 is vertical, the choice of probe wavelength determines the OCR on the PES. For instance, if $\lambda_2 = \lambda_2^\infty$, then the OCR will extend from some finite internuclear separation (whose value depends on the spectral bandwidth of the probe) to infinity. As the probe wavelength is increased, the OCR moves towards smaller internuclear separation for the case under consideration (red shift). Since the FTS experiment is a clock of the time required to move from R_0 to the OCR, the transient must therefore reach its maximum at successively shorter times as the wavelength is tuned further to the red.[32]

This last point is important, as it shows that the detuning experiments can be interpreted as a generalization of the clocking technique. For each probe wavelength, the value of t^* has been determined. If the location of the probe OCR [i.e., $R(t^*)$] is known for that pulse, then the average velocity required to reach there is deduced. The more general situation, in which the PES is determined without such knowledge, is discussed below.

The transition state $[\mathrm{I}\cdots\mathrm{CN}]^{\ddagger *}$, persists only for a finite time, and its duration is measured in this detuning experiment. In Fig. 7, for example, the deconvolved absorption persists only for $\sim$20–50 fs. According to the model, the

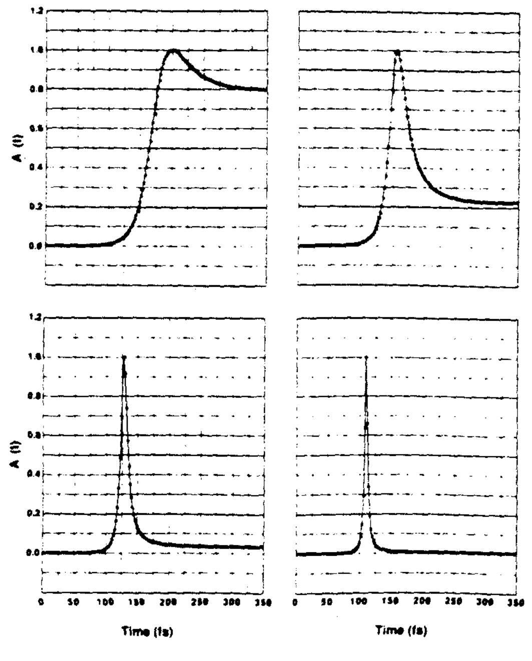


FIG. 7. Deconvolved fits to the detuned experiment results. The fits presented in Fig. 5 are shown with the experimental response deconvolved. Fit values are: (a) $L = 0.84$ Å, $t^* = 200$ fs, $\gamma = 96.5$ cm^{-1}; (b) $L = 0.84$ Å, $t^* = 156$ fs, $\gamma = 102$ cm^{-1}; (c) $L = 0.84$ Å, $t^* = 128$ fs, $\gamma = 83.5$ cm^{-1}; and (d) $L = 0.88$ Å, $t^* = 110$ fs, $\gamma = 64$ cm^{-1}.

time required to traverse the full width of the absorption at half-maximum is

$$\tau^* \equiv 2(t - t^*) \simeq \sqrt{2f} L_1^*/v^* , \qquad (6)$$

where f is in this case ~ 1, and v^* is the speed of separation at $R = R^*$. The length factor L_1^* (the length in which the potential changes by γ at the transition-state probing) is simply $\gamma L_1/V_1(R^*)$. The quantity $V_1(R^*)/L_1$ is the rate of change of $V_1(R)$ with distance at $R = R^*$. To obtain transition-state "lifetimes" of the order of 50 fs, this suggests that the potential dropped to a hundred wave numbers or so, as shown below.

C. Tuning to the blue

The LIF spectrum of free-CN consists not of a single transition at 388.5 nm, but exhibits the many transitions of the P and R branches. For 308 nm excitation, the R branch is fully resolved and extends to 381.5 nm, as shown by Fisher *et al.*[34] (In addition to $v = 0$, the ground-state vibrational level, rotational lines, much weaker lines due to $v = 1$ were also reported.) These transitions to the blue of the bandhead are associated with different rotational quantum numbers of the CN fragment (up to ~ 50), and as mentioned in the Introduction have been analyzed to give product state distributions.

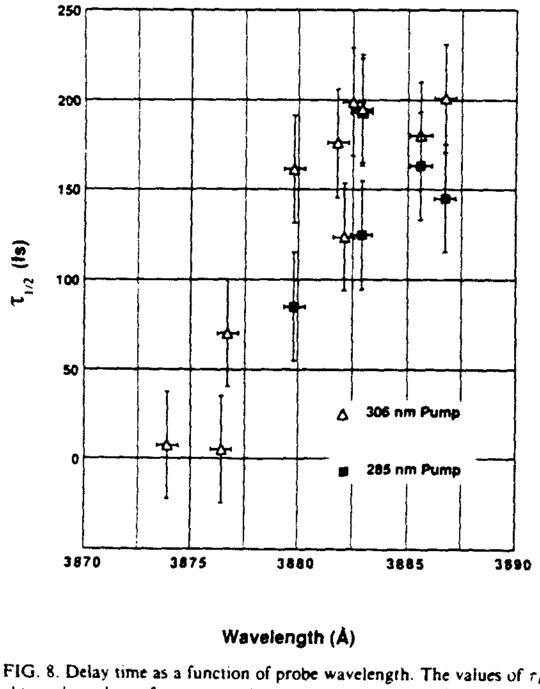


FIG. 8. Delay time as a function of probe wavelength. The values of $\tau_{1/2}$ obtained are shown for an approximate pump wavelength of 306 nm (open triangles) and 285 nm (solid squares). The FWHM of the probe was ~ 60 cm^{-1} in each case.

The spectral width of the femtosecond probe is broad, so that the excitation of a single rotational level is impossible. However, we have attempted to selectively probe regions within the P branch of the CN spectrum which are associated with either high or low rotational quantum states. For example, the region near the bandhead is associated with high rotational states ($N'' \geqslant 20$), and near the start of the P branch at 387.5 nm, with lower quantum numbers.

The clocking experiment was performed at various probe wavelengths between 388.7 and 387.4 nm. The values of $\tau_{1/2}$ determined from these measurements are shown in Fig. 8, for both 306 and 285 nm pump wavelengths. For pulses at the bandhead, as previously discussed, this experiment yields a time delay of $\tau_{1/2} \sim 200$ fs. However, a substantial decrease in $\tau_{1/2}$ was observed as the probe was shifted towards shorter wavelengths.[35]

An obvious possible cause for this observation is due to the change in the translational energy with rotational excitation of the CN. This is illustrated in Fig. 9. For 7000 cm^{-1} of total available energy, the low N'' states will be produced with essentially all this energy in translation. However, for high N'' states the translational energy will be less than the available energy; for $N'' = 25$, $E_R = 1250$ cm^{-1}, and the translational energy becomes 5750 cm^{-1} (vibrational excitation is assumed to be negligible[34]). In the center-of-mass frame, v is proportional to $E^{1/2}$, and $\tau_{1/2}$ at most should change from ~ 200 to ~ 180 fs. This is a much smaller effect than the data in Fig. 8 shows.

Probing the different N'' states of the CN is sensitive to a more interesting dependence, namely, the angular part of the potential. In general, the PES is not only radial but it has an angular dependence described by the angle (θ) between I and the center-of-mass of the CN. For ICN, the potential has been postulated and used to fit the product state distribution, e.g., as done by Goldfield *et al.*[12] A form that is discussed by Schinke[36] and others is generally written as

$$V(R,\theta) = A \exp\left[-\frac{R - R_0}{L} \right] \exp\left[-\frac{\epsilon(\cos^2\theta - 1)}{L} \right], \qquad (7)$$

where ϵ is a measure of the coupling between the dissociation coordinate R and θ. When $\theta = 0$, we recover the single radial potential (isotropic). But for negative ϵ, we have a bent ex-

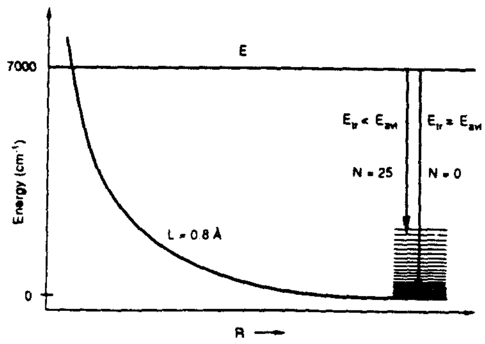


FIG. 9. Change in translational energy with rotational energy of CN. The probe pulse either samples regions of high or low rotational quantum numbers.

cited state which has a minimum at $\theta = 90°$ and a maximum at $\theta = 0$. Dugan[16] has discussed this bending (of CN relative to I) at the 308 nm excitation and obtained a bending angle of $\theta_0 = 18°$. Basically, if the molecule were linear ($\theta_0 = 0$, on the average), then the rotational distribution would show a maximum at $N'' \approx 0$; instead, the maximum seen in these experiments is at higher N'' values for this wavelength of excitation. As he pointed out, at these long wavelengths there is one or two quanta of bending vibration (ground state), and their classical turning points are at 14° and 19°. We will not attempt quantification of the angle here, but the important points are that the bent state produces a range of torques and that there is time evolution for N.

Elsewhere,[37] we will show that such bending may describe the observation. At shorter times the rotational distribution is biased toward lower N states, while at longer times all states are produced. Because of bending and the centrifugal potential, probing higher N states of the distribution will cause a longer delay than the lower N states. Such changes should be directly related to $V(R,\theta)$ and simulations are underway to quantify this effect and consider other channels. It should be noted that the clock used here to measure $\tau_{1/2}$ is sensitive to the separation of the I and CN (and any changes in the CN bond length). Even though the bending may develop at earlier times compared to the recoil times, it has a signature on the different angular momentum states of CN, and the total time it will take a trajectory to make it to a given N state will depend on $V(R,\theta)$.

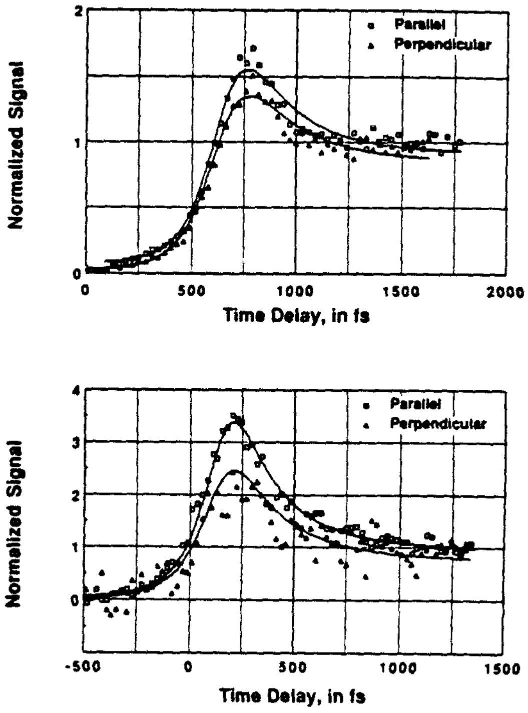


FIG. 10. Results of femtosecond alignment experiments. FTS data taken off-resonance, with either parallel (open squares) or perpendicularly polarized (solid triangles) pump and probe beams. The probe polarization was kept parallel to the detection axis. Two data sets are shown, taken under different experimental conditions. Note that the time delay in the upper curve is arbitrary (the zero-of-time was not determined). The data is normalized to a value of unity at long time delays. The lines are guides to the eye.

D. Femtosecond coherence and alignment

For these experiments, we used a polarized femtosecond pulse arrangement to probe the degree of coherence and alignment as a function of time. Figures 10(a) and 10(b) each show two data sets, taken under identical conditions, one with parallel, and the other with perpendicular polarizations of the pump and probe beams. The data has been normalized to an asymptotic value of one, and (for the lower data set only) the absolute zero-of-time has been determined. The data on the top set is for a bluer wavelength of probing, and was done under entirely different conditions.

The characteristic results of these femtosecond alignment experiments are as follows: (1) the parallel polarization transient achieves a larger normalized amplitude in the first hundred femtoseconds or so after pump excitation, (2) the general shape of the transient is similar, and (3) the time delays at which the signals achieve their maximum values are equal (to within experimental resolution). The change in amplitude for $A_{\parallel}(t)$ and $A_{\perp}(t)$ is not very large but reproducible in these two different sets of experimental results. It is possible that when we complete these measurements using shorter pulses that we will be able to resolve the anisotropy $A_{\parallel}(t) - A_{\perp}(t)$ better.[38]

As the CN separates from the I atom, the CN rotational states evolve in a "coherent packet" which then dephases to an incoherent product state population. The time evolution for this process determines the degree of alignment $[A_{\parallel}(t)$ and $A_{\perp}(t)]$ which depend on $P_2(N\omega_0 t)$, where P_2 is the second Legendre polynomial. If the CN is produced in, for instance, a few very low N states, then the coherence time will be on the order of the inverse of the rotational constant; it becomes shorter for larger N. In this case, beats (with in-phase and out-of-phase characteristics) for the two polarizations will be observed. On the other hand, if the CN is produced in a distribution of N states, then dephasing of the initial coherence will result in an initial decay and some recurrences.[24] For the problem at hand, taking the dissociation as an impulse to excite the CN coherently with a distribution of rotational energies, the coherence time in the course of dissociation can be estimated by[37]

$$\tau_c \approx \frac{2.2}{\sqrt{B\langle E_R\rangle}}, \tag{8}$$

where τ_c is in picoseconds and both the constant B and the average rotational energy $\langle E_R\rangle$ are in cm^{-1}. At our excitation wavelength, this gives $\tau_c \approx 44$ fs.

As discussed elsewhere,[37] one can write expressions for $A_{\parallel}(t)$ and $A_{\perp}(t)$ which include explicitly τ_c, and show the relative importance of τ_c to τ^* and t^*. If the coherence time is long compared to t^*, then both $A_{\parallel}(t)$ and $A_{\perp}(t)$ should have similar shapes and will only differ by a constant scale factor of 3. However, if the coherence time is sufficiently short relative to our pulse, the initial part of the transient will be convolved. The similarity of the transient is consistent with this picture, and our observations, therefore, suggest that the coherence time is on the time scale of our experiment. Shorter pulses will help resolve this effect and obtain accurate T_c. The shorter the t^* of observation, the larger the anistropy should be, consistent with the preliminary data in Fig. 10. (Note the values of the asymptotes and the initial anisotropy.) Wilson's group is proposing this type of anisotropy experiments to probe changes in the dynamics of the potential ongoing from the gas phase to the condensed phase.[39]

E. The potential energy surface

From knowledge of the length parameter L obtained above, we can now deduce the change of the potential with time $V(t)$ and the corresponding $V(R)$. We will use the same approximation invoked in Sec. III B. The experimentally measured $A(t)$ can be used to obtain the explicit form of $V_1(t)$:

$$V_1(t) = V_1(t^*) \pm \gamma \sqrt{\frac{A_{\max}}{A(t)} - 1}\,, \qquad (9)$$

where $A_{\max}$ is a constant (given by C/γ^2 in I). The $+$ ($-$) sign is used for $t <$ ($>$) t^*. For an exponential potential in R, this $V(t)$ will be explicitly given by

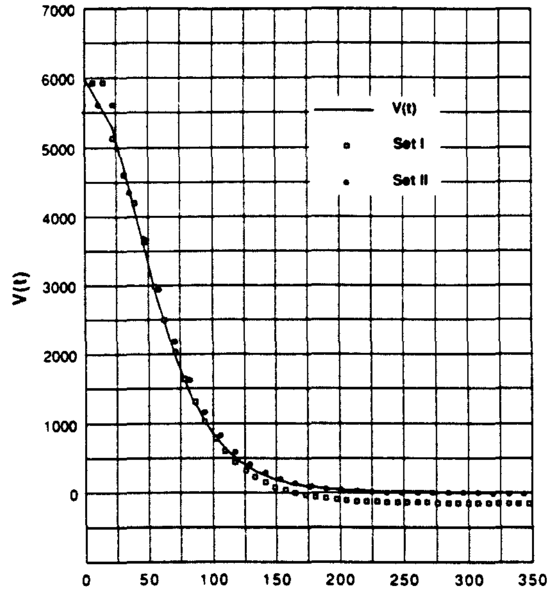


FIG. 12. Inversion to the potential. The calculation of the potential as a function of time, $V(t)$, is shown for the data of Fig. 11. The solid line is the theoretical result [Eq. (10)] using the fit value for L_1.

$$V_1(t) = E\,\text{sech}^2(vt/2L_1)\,. \qquad (10)$$

Figure 11 shows two (deconvolved) sets of experimental transients, $A(t)$ vs t. Using Eq. (9), we have inverted to obtain $V_1(t)$ (Fig. 12), and as expected we obtained the presumed potential, which has dropped down to a negligible value in ~200 fs. It is important to note that the shape of $V_1(t)$ vs t, especially at short times, is not exponential (see also Fig. 13). This is because the recoil velocity changes with

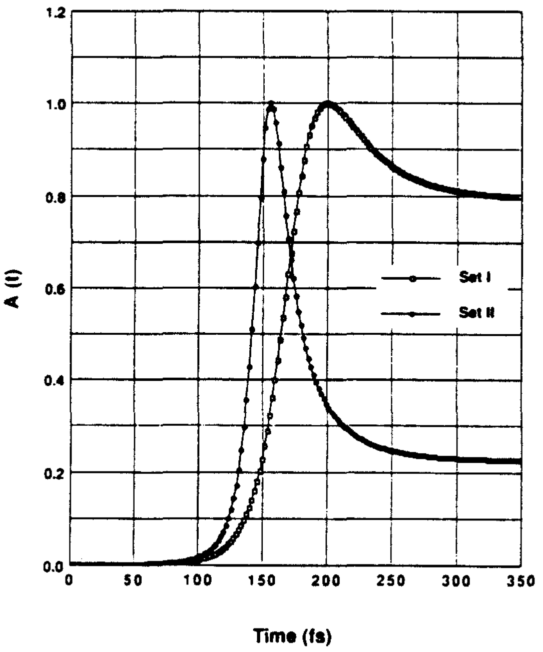


FIG. 11. Deconvolved absorption from detuning experiment. Data taken from Figs. 7(a) (open squares) and 7(b) (solid circles). Note the time shift between the two transients. The two transients will be used for the inversion shown in Fig. 12.

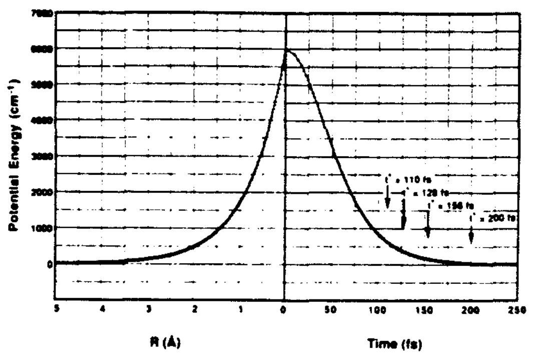


FIG. 13. The potential as a function of time and length. V is shown as a function of the internuclear separation (left) and time (right). The values at left show an exponential shape with the experimentally deduced length parameter, and those at right are derived from Eq. (10). The values of t^* from the fits of the data of Fig. 5 are indicated.

time; at early times the fragments separate slowly, but eventually accelerate to the terminal velocity. For this same potential we display in Fig. 13 the corresponding $V_1(R)$ and t^* for four sets of probes. This helps illustrate the corresponding distance scales involved.

One drawback of the above procedure is the assumption made about the upper surface. Naturally, we wish to invert to the potential without any *a priori* assumptions about the shape of V_2 (exponential repulsion, R^{-2}, etc.). There is a method by Bernstein and Zewail[33] which inverts to give $\Delta V(t)$ vs t (i.e., the change in the difference potential with time) and, by pump-tuning experiments, $V_1(t)$. This method will be described in Ref. 33 and we briefly mention it here to make a connection with the above discussion. The idea is as follows: For a given t^*, $A(t)$ is known as a function of t. Also, $A(\Delta)$ is known, where Δ is the detuning from λ_2^∞ and $A(\Delta)$ is a measure of the amount of absorption of the free fragment as a result of the finite width of the pulse. Therefore, by projection of $A(\Delta)$ vs Δ onto $A(t)$ vs t, we obtain ΔV vs t without *a priori* assumptions on the shape. In the future, we expect two improvements in order to construct the PES over a larger range of R: the detuning should be extended, to values of λ_2^* further to the red (and blue), and the pulse widths should be narrowed. Such experiments are feasible.

A final note should be made about the quantum mechanical simulation of the ICN reaction. As mentioned in paper I, Williams and Imre[40] have simulated the wave packet for dissociation of ICN (see Fig. 16 of I) as a function of detuning. Their results are in good agreement with the experimental shape of the transients (see Fig. 17 in I). In their simulation, they used $\alpha = 1/L_1 = (0.33\ \text{Å})^{-1}$, and they simulate the different experiments at different λ_2^* (388.9, 389.5, and 390.5 nm). Note the consistency in the time scale, time shift, and the magnitude of the asymptotes with the experiments. When α is rescaled to agree with our experimentally deduced value of $(0.8\ \text{Å})^{-1}$, the expected time delay (or wave packet displacement to dissociation) is close to our $\tau_{1/2}$. It is pleasing to see that the quantum wave packet model is reproducing the experimental features very well, and is in agreement with the classical predictions. There are, of course, many details that have not been quantified yet. We are currently examining details of the PES and their influence on the shape and the movement of the packet, under different experimental conditions of interest.

IV. CONCLUSIONS

In this study, we have applied FTS to the dissociation reaction of ICN. The breaking of the I–CN bond has been clocked and the characteristic time $\tau_{1/2}$ has been measured. Short-lived transition states of the reaction ($[\text{I}\cdots\text{CN}]^{\ddagger*}$) were observed in experiments where the probe pulses were detuned to longer wavelength than the bandhead of the free-CN transition. Both the on-resonance (clocking) and off-resonant (buildup and decay) behaviors are reproduced theoretically using classical and quantum mechanical models of FTS presented previously in paper I. From the experimental observations and using an exponential repulsion, the length scale for the PES of ICN was deduced for the range of R studied. Tuning the probe frequency to the blue yielded smaller values of $\tau_{1/2}$, which we relate to the angular part of the PES, and can be explained by the consideration of the femtosecond time evolution of the different rotational quantum numbers. Femtosecond alignment studies of ICN, as well as theory, suggest that coherence during dissociation is lost rapidly, estimated to be ~44 fs.

This paper gives a methodology for treating these first FTS studies of reactions, and work is continuing on ICN and other reactions to invert the FTS data and obtain more accurate PES. We limited ourselves here to simple descriptions, but there are a number of theoretical and experimental issues to be further developed. As we gain more experience in this field, we hope to exploit the full capabilities of these FTS techniques and learn more about real-time femtosecond dynamics (population and alignment) of elementary reactions.

ACKNOWLEDGMENTS

This work was supported by AFOSR (Grant No. 87-0071). Contribution No. 7814. We wish to thank the referee of this and the previous paper for a thorough and comprehensive review.

[1] M. J. Rosker, M. Dantus, and A. H. Zewail, J. Chem. Phys. **89**, 6113 (1988).
[2] M. Dantus, M. J. Rosker, and A. H. Zewail, J. Chem. Phys. **87**, 2395 (1987).
[3] G. W. King and A. W. Richardson, J. Mol. Spectrosc. **21**, 339 (1966); see also Ref. 8(b).
[4] H. Okabe, *Photochemistry of Small Molecules* (Wiley–Interscience, New York, 1978).
[5] A. P. Baronavski and J. R. McDonald, Chem. Phys. Lett. **45**, 172 (1977).
[6] J. H. Ling and K. R. Wilson, J. Chem. Phys. **63**, 101 (1975).
[7] (a) W. Krieger, J. Hager, and J. Pfab, Chem. Phys. Lett. **85**, 69 (1982); (b) W. H. Fisher, T. Carrington, S. V. Filseth, C. M. Sadowski, and C. H. Dugan, Chem. Phys. **82**, 443 (1983); (c) W. H. Fisher, R. Eng, T. Carrington, C. H. Dugan, S. V. Filseth, and C. M. Sadowski, *ibid.* **89**, 457 (1984); (d) W. J. Marinelli, N. Sivakumar, and P. L. Houston, J. Phys. Chem. **88**, 6685 (1984); (e) I. Nadler, H. Reisler, and C. Wittig, Chem. Phys. Lett. **103**, 451 (1984); (f) G. E. Hall, N. Sivakumar, and P. L. Houston, J. Chem. Phys. **84**, 2120 (1986); (g) see also Refs. 5 and 9. (h) M. J. Sabety-Dzvonik and R. J. Cody, J. Chem. Phys. **66**, 125 (1977).
[8] (a) W. M. Pitts and A. P. Baronavski, Chem. Phys. Lett. **71**, 395 (1980); (b) W. P. Hess and S. R. Leone, J. Chem. Phys. **86**, 3773 (1987).
[9] I. Nadler, D. Mahgerefteh, H. Reisler, and C. Wittig, J. Chem. Phys. **82**, 3885 (1985).
[10] (a) M. A. O'Halloran, H. Joswig, and R. N. Zare, J. Chem. Phys. **87**, 303 (1987); E. Hasselbrink, J. R. Waldeck, and R. N. Zare (to be published); (b) see Ref. 7(f).
[11] (a) M. D. Morse, K. F. Freed, and Y. B. Band, J. Chem. Phys. **70**, 3620 (1979); (b) M. N. R. Ashfold, M. T. Macpherson, and J. P. Simons, Top. Curr. Chem. **86**, 1 (1979); J. P. Simons, in *Gas Kinetics and Energy Transfer*, edited by P. G. Ashmore and R. J. Donovan (Chemical Society, London, 1977), Vol. 2, p. 58; (c) see also Ref. 9.
[12] E. M. Goldfield, P. L. Houston, and G. S. Ezra, J. Chem. Phys. **84**, 3120 (1986).
[13] (a) K. E. Holdy, L. C. Klotz, and K. R. Wilson, J. Chem. Phys. **52**, 4588 (1970); (b) J. A. Beswick and J. Jortner, Chem. Phys. **24**, 1 (1977); (c) U. Halavee and M. Shapiro, *ibid.* **21**, 105 (1977); (d) J. A. Beswick and W. M. Gelbart, J. Phys. Chem. **84**, 3148 (1980); (e) M. D. Pattengill, Chem. Phys. **78**, 229 (1983); **87**, 419 (1984); (f) B. A. Waite, H. Helvajian, B. I. Dunlap, and A. P. Baronavski, Chem. Phys. Lett. **111**, 544 (1984); (g) see also Refs. 7(d) and 11(a).
[14] N. F. Scherer, J. L. Knee, D. D. Smith, and A. H. Zewail, J. Phys. Chem. **89**, 5141 (1985).
[15] (a) A. P. Baronavski, Chem. Phys. **66**, 217 (1982); (b) see Refs. 7(b) and 7(c).

[16](a) C. H. Dugan and D. Anthony, J. Phys. Chem. **91**, 3929 (1987); (b) C. H. Dugan, *ibid.* **92**, 720 (1988).

[17](a) C. H. Greene and R. N. Zare, Annu. Rev. Phys. Chem. **33**, 119 (1982), and references therein; (b) R. Bersohn, Isr. J. Chem. **14**, 111 (1975).

[18](a) R. N. Zare, Mol. Photochem. **4**, 1 (1972); (b) S. Yang and R. Bersohn, J. Chem. Phys. **61**, 4400 (1974); (c) C. Jonah, *ibid.* **55**, 1915 (1971); (d) R. Schmiedl, H. Dugan, W. Meier, and K. H. Welge, Z. Phys. A **304**, 137 (1982); (e) P. Andersen, G. S. Ondrey, B. Titze, and E. W. Rothe, J. Chem. Phys. **80**, 2548 (1984).

[19]β is a product of three functions; that is: $\beta = 2P_2(\cos\gamma) P_2(\cos\alpha) f(\langle\omega\rangle\langle\tau\rangle)$, where γ is the angle between the transition moment and the molecular axis, α is the angle between the recoil direction and the molecular axis, and $f = [1 + (\langle\omega\rangle\langle\tau\rangle)^2]/[1 + 4(\langle\omega\rangle\langle\tau\rangle)^2]$ describes the effect of the parent rotation on the angular distribution. For axial recoil and prompt dissociation, $\beta = 2$ for the parallel transition and $\beta = -1$ for the perpendicular transition.

[20]See Ref. 7(d).

[21]The only adjustment made was the attenuation of the probe beam, by the translation of a variably coated neutral density filter (Al coating on thin quartz substrate). We have verified that this translation does not affect the $t = 0$ location, to within our experimental resolution (< 10 fs).

[22](a) H. Tsubomura and T. Sakata, Chem. Phys. Lett. **21**, 511 (1973); K. Fuke and S. Nagakura, J. Mol. Spectrosc. **64**, 139 (1977); (b) J. R. Woodworth, T. A. Green, and C. A. Frost, J. Appl. Phys. **57**, 1648 (1985).

[23]P. M. Felker and A. H. Zewail, Adv. Chem. Phys. **70**, 265 (1988), and references therein.

[24]P. M. Felker and A. H. Zewail, J. Chem. Phys. **86**, 2460 (1987); J. S. Baskin, P. M. Felker, and A. H. Zewail, *ibid.* **86**, 2483 (1987).

[25](a) C. K. Luk and R. Bersohn, J. Chem. Phys. **58**, 2153 (1973); (b) W. M. Jackson, *ibid.* **61**, 4177 (1974); (c) T. J. Cook and D. H. Levy, *ibid.* **57**, 5059 (1972).

[26]See, for example: (a) R. Vasudev, R. N. Zare, and R. N. Dixon, J. Chem. Phys. **80**, 4863 (1984); (b) P. L. Houston, J. Phys. Chem. **91** (Special Issue of Dynamical Stereochemistry), 5388 (1987); (c) J. Simons, *ibid*, **91**, 5378 (1987).

[27](a) T. F. Heinz, S. L. Palfrey, and K. B. Eisenthal, Opt. Lett. **9**, 359 (1984); S. L. Palfrey and T. F. Heinz, J. Opt. Soc. Am. B **2**, 674 (1985); (b) M. W. Balk and G. R. Fleming, J. Chem. Phys. **83**, 4300 (1985); (c) R. Trebino, C. E. Barker, and A. E. Siegman, IEEE J. Quantum Electron. **QE-22**, 1413 (1986); (d) P. C. Becker, R. L. Fork, C. H. Brito Cruz, J. P. Gordon, and C. V. Shank, Phys. Rev. Lett. **60**, 2462 (1988).

[28]The available energy is calculated as follows: We take the total energy for the 306 nm excitation and subtract the bond energy ~74 kcal/mol (see, e.g., Ref. 12 and references therein). When subtracting the average rotational energy of the CN (1250 cm^{-1}) from this available energy, we obtain the translational energy.

[29]R. Bersohn and A. H. Zewail, Ber. Bunsenges. Phys. Chem. **92**, 373 (1988).

[30](a) T. S. Rose, M. J. Rosker, and A. H. Zewail, J. Chem. Phys. **88**, 6672 (1988); (b) M. J. Rosker, T. S. Rose, and A. H. Zewail, Chem. Phys. Lett. **146**, 175 (1988).

[31]See Ref. 13 [e.g., (a) and (c)] for extraction of $L^{-1} = \alpha$ from the absorption spectrum.

[32]We assume no van der Waals well or switching in shape between V_1 and V_2. This is discussed in detail in Ref. 33.

[33]R. B. Bernstein and A. H. Zewail, J. Chem. Phys. (submitted).

[34]See Fig. 7 of Ref. 7(b).

[35]In the experiments, the probe tuning procedure had the effect of also changing the pump wavelength, in the range from 305 to 308 nm. However, the additional energy available to the fragments as the pump frequency varied was negligible, and was not believed to affect $\tau_{1/2}$ significantly. Evidence for this assertion can be seen in the results obtained with a 285 nm pump wavelength, where $\tau_{1/2}$ changed only to 160 fs.

[36]R. Schinke, J. Phys. Chem. **92**, 3195 (1988), and references therein.

[37]A. H. Zewail, Faraday Discuss. (to be published).

[38]Normalizing the $A_{\parallel}$ and $A_{\perp}$ signals on an absolute scale is experimentally difficult because of delicate changes in the optical alignment when rotating the half-wave plate. We use the long time asymptote as a marker to see the change in the time dependence of the signals. In future experiments, we will attempt to normalize the data more accurately and to use magic angle methods. (See Ref. 24.)

[39]K. Wilson (private communication).

[40]S. O. Williams and D. G. Imre, J. Phys. Chem. (in press).

Chapter IV

Applications of FTS

2. Femtosecond real-time probing of reactions. V. The reaction of IHgI

Femtosecond real-time probing of reactions. V. The reaction of IHgI

M. Dantus, R. M. Bowman, M. Gruebele, and A. H. Zewail
Arthur Amos Noyes Laboratory of Chemical Physics,[a] *California Institute of Technology, Pasadena, California 91125*

(Received 30 June 1989; accepted 15 August 1989)

The dissociation reaction of HgI_2 is examined experimentally using femtosecond transition-state spectroscopy (FTS). The reaction involves symmetric and antisymmetric coordinates and the transition-state is well-defined: $\mathrm{IHgI}^* \to [\mathrm{IHgI}]^{\ddagger *}_{Q_s, Q_a} \to \mathrm{HgI} + \mathrm{I}$. FTS is developed for this class of ABA-type reactions and recurrences are observed for the vibrating fragments (symmetric coordinate) along the reaction coordinate (antisymmetric coordinate). The translational motion is also observed as a "delay time" of the free fragments. Analysis of our FTS results indicates that the reaction wave packet proceeds through two pathways, yielding either $\mathrm{I}(^2P_{3/2})$ or $\mathrm{I}^*(^2P_{1/2})$ as one of the final products. Dissociation into these two pathways leads to HgI fragments with different vibrational energy, resulting in distinct trajectories. Hence, oscillatory behaviors of different periods in the FTS transients are observed depending on the channel probed ($\sim$300 fs to $\sim$1 ps). These results are analyzed using the standard FTS description, and by classical trajectory calculations performed on model potentials which include the two degrees of freedom of the reaction. Quantum calculations of the expected fluorescence of the fragment are also performed and are in excellent agreement with experiments.

I. INTRODUCTION

Femtosecond transition-state spectroscopy (FTS)[1,2] has been applied in this series of papers[3–6] to a number of chemical reactions of the type

$$\mathrm{AB}^* \to [\mathrm{AB}]^{\ddagger *} \to \mathrm{A} + \mathrm{B} \tag{1}$$

as in the reactions of alkali metal halides[6,7] and

$$\mathrm{ABC}^* \to [\mathrm{ABC}]^{\ddagger *} \to \mathrm{AB} + \mathrm{C} \tag{2}$$

as in the reaction of ICN.[3,4] More recently we have reported femtochemistry studies of the reaction of IHgI[8] (i.e., ABA), where the motion along two coordinates was observed. In this paper (V) new FTS results are presented for this reaction, and comparison with theory is made. These studies of the femtosecond dynamics in ABA-type reactions help us develop FTS in systems with more than one nuclear coordinate.

The HgI_2 system is attractive for several reasons. Firstly, it is much different from the case of ICN.[3,4] For ICN, the motion between the carbon and nitrogen atoms is considered frozen, allowing one to treat the ICN as a quasidiatomic. In the case of HgI_2, motion is expected between both iodine atoms and the mercury atom adding an additional coordinate on the potential energy surface (PES). Secondly, the equivalence of the two bonds make the PES symmetric with respect to the I, HgI coordinate and the reaction path is well defined along the antisymmetric coordinate (see Fig. 1). Thirdly, because of the change in the structure in going from the transition-state to the final products, one expects changes in vibrational excitation of HgI fragments produced from different starting conditions. Detection of the laser-induced fluorescence (LIF) at different wavelengths gives us a handle on the different vibrational states produced in the reaction. Fourthly, the bending motion leads to rotational excitation in the final products and this can be probed in real time, as reported recently.[8(b)] Finally, a great deal of information is available on the spectroscopy,[9] product-state distributions,[10(a)] and the half-[10,11] and full-collision ($\mathrm{Hg} + \mathrm{I}_2 \leftrightarrow \mathrm{HgI} + \mathrm{I}$)[12] dynamics. Molecular beam studies by the group of Bernstein[12] and theoretical calculations by Mayer *et al.*,[12] have provided the foundation for describing the PES of the ground state.

The dissociation of HgI_2 has been studied[9,10] and several observations have been made. Excitation at 310 nm most

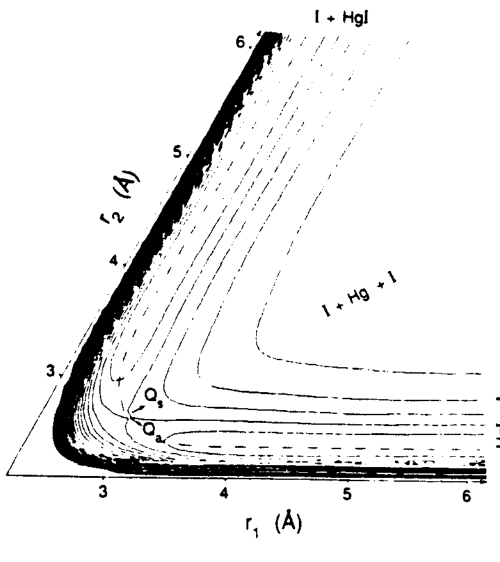


FIG. 1. Schematic of the mass-weighted PES for the reaction of linear HgI_2. The direction for the symmetric (Q_s) and antisymmetric (Q_a) stretches are indicated with arrows.

[a] Contribution No. 7971.

importantly accesses three dissociative channels of HgI_2[10]:

$$HgI_2 + h\nu \rightarrow HgI(X\,^2\Sigma) + I(^2P_{3/2}), \quad (3a)$$
$$\rightarrow HgI(X\,^2\Sigma) + I^*(^2P_{1/2}), \quad (3b)$$
$$\rightarrow Hg + I + I, \quad (3c)$$

producing iodine atoms in either the ground spin–orbit state $I(^2P_{3/2})$ or in the excited state $I^*(^2P_{1/2})$. At this wavelength of excitation it is known that the channel leading to ground state iodine [Eq. 3(a)] is preferred over the I^* channel [Eq. 3(b)] by about a factor of 4.[10(a)] The HgI fragment from both channels is produced in the ground electronic state ($X\,^2\Sigma$). FTS studies reported here are based on probing the HgI fragment during the dissociation and taking it to a second PES leading to the production of fluorescent HgI ($B\,^2\Sigma$). Due to the large change in the bond lengths from HgI ($X\,^2\Sigma$) to the excited state HgI ($B\,^2\Sigma$) the absorption and fluorescence spectra are very dispersed. Fortunately the spectroscopy has been analyzed by Cool *et al.*,[10(b)] and the Franck–Condon factors and assignments of these spectra are known.

The FTS analysis follows the guidelines described in the first paper of the series[3]; the new feature is the analysis of the fluorescence following the probing the transition states. Typically, the FTS signal involves the detection of the LIF generated by the probe laser. Probing of the reaction in real time can be achieved by tuning λ_2 (the probe laser) to a transition of the free fragment (λ_2^∞ detection) or when it is tuned to transitions originating from transition configurations (λ_2^* detection). The fluorescence spectrum of HgI ($B\,^2\Sigma$), being well known, provides a unique opportunity for deducing which vibrational levels of the ground state HgI are accessed by the dissociation and which ones are accessed by the probe. Fixing the probe wavelength λ_2^*, and subsequently detecting the LIF at different wavelengths in a sense gives us the opportunity of discriminating the dynamics to a restricted set of trajectories. This new feature is exploited in this paper in order to map the PES for the HgI_2 dissociation reaction by the analysis of different trajectories observed in real time.

The organization of the paper is as follows: In Sec. II, we describe the experimental details specific to the FTS of HgI_2. This includes a discussion of a number of characterization and diagnostic experiments that we have performed. In Sec. III, we summarize the results. In Sec. IV the results are first discussed in terms of the simple FTS picture and then a comparison is made with classical two-dimensional trajectory calculations. Finally, a quantum mechanical analysis of the HgI fluorescence is performed and is used to deduce the experimental observations.

II. EXPERIMENTAL

The experimental methodology of FTS has been described in other femtochemistry studies.[3,4] In this section, we provide specific details pertinent to the FTS experiments of the HgI_2 reaction.

A. FTS experiments

Briefly, the femtosecond pulses were generated from a colliding pulse mode-locked ring dye laser (CPM),[13] and were amplified by a Nd:YAG-pumped pulsed dye amplifier (PDA).[14] The output pulses, as short as 50 fs duration, had 0.3–0.5 mJ of energy and 20 Hz repetition rate.

The pump wavelength (310 nm) was the second harmonic of the PDA output. Second harmonic generation was achieved in a thin (0.5 mm) KD*P phase-matched crystal. Two probe wavelengths were used in this studies. The output of the PDA at 620 nm was used without further conversion as a probe wavelength for experiments requiring tuning far off resonance of the free fragment transition. The probe pulses closer to the resonance transitions of the HgI ($X \leftrightarrow B$) states were generated by mixing the remainder of the PDA output with 1.06 μm light from the Nd:YAG laser in a thin (0.5 mm) KD*P crystal resulting in light at 390 nm. For experiments with 620 nm probing the pulses were as short as 50 fs, for $\lambda_1 = 310$ nm and $\lambda_2 = 390$ nm experiments we recorded cross correlations of 80 to 90 fs. In all cases care were exercised to insure that the pulses were chirp free (transform limited), so that $\Delta\nu\Delta t = 0.3$ to 0.5.

The pump and probe beams, with proper attenuation and parallel or perpendicular polarization, were delayed in time relative to one another in a Michelson interferometer, and were collinearly recombined and focused into the reaction chamber. Two custom made cells were used in these experiments. The first cell allows controlled flow of the HgI_2 and can be fitted with an electrode which we have used for determination of the absolute zero of time *in situ* by the DEA/MPI technique.[3,4] The second cell is a small quartz cylinder in which a small amount of HgI_2 is sealed in vacuum. Due to the fast recombination of I and HgI[9(a)] in the cell we confirmed that vapor flow was not necessary for our experiments. The static cell was used in most of the experiments because of its convenience.

LIF of the HgI fragment was collected at right angles to the direction of the pump/probe pulses through a $f/1.5$ lens arrangement. Spurious light and scattered light were rejected by dispersing the LIF through a 0.34 m monochromator. The LIF was collected at right angles by a photomultiplier tube (PMT), and a boxcar integrator averaged a portion of the PMT voltage. The entire experiment was controlled by a microprocessor, which adjusted the optical delay line of the Michelson interferometer and acquired and numerically averaged the boxcar output. Dispersed LIF spectra were acquired by a computer driven monochromator with the probe at a fixed delay with respect to the pump pulses. The fluorescence spectra were not corrected for laser intensity or PMT spectral response.

Characterization of the pump and probe pulses was performed for each data set. The characterization consisted of taking the spectrum of each of the pulses and measuring the time duration of the pulses. For $\lambda_1 = 310$ nm, $\lambda_2 = 390$ nm experiments, an autocorrelation of the PDA output in a noncollinear autocorrelator determined the temporal characteristics of the pulses. For 310/390 experiments a cross correlation technique using difference frequency generation in a nonlinear crystal ($\omega_{DFG} = \omega_{pump} - \omega_{probe}$) was used, the light near 1.45 μm was detected with a germanium photodiode. The overall temporal response function of the apparatus was determined using the DEA/MPI technique.

B. Diagnostic experiments

The characterization of the HgI_2 reaction signal involved numerous diagnostic experiments. First, to establish the linearity of the signal with respect to the intensity of both the pump and the probe laser beams, the dependence of the LIF signal on the pump or probe energy was recorded. This measurement was made for transients resulting from $\lambda_1 = 310$ nm and $\lambda_2 = 390$ nm or 620 nm. The results are plotted on a log–log scale in Fig. 2. For each case the data falls on a line with a slope near unity, which is evidence for the linearity of the FTS experiment as well as the detection apparatus. As a further test, FTS transients for both probe wavelengths were taken under high and low intensity of both pump and probe beams. We observed no evidence of saturation in any of these measurements. The signal level is consistent with previous FTS experiments carried out in this laboratory taking into account various factors such as cross sections of the transitions, detector sensitivity and quantum yields.

In order to insure that the signal is due to the HgI_2 monomer and not polymers, and to determine the extent of collisional quenching of the signal we performed an experiment that compares the signal intensity to the theoretically calculated number density at different temperatures. The results of this experiment are presented in Fig. 3. The signal was found to closely follow the number density curve. Plotting the number density vs the signal [Fig. 3(b)], gives a straight line with a slope of unity indicating that in the range of pressures and temperatures used for these experiments the signal was not affected by collision quenching and it did not depend on dimers or polymers. All the data reported here was acquired in conditions far from the extremes used for these diagnostic experiments.

The pump beam (310 nm) was found to produce some background HgI fluorescence. A pump intensity dependence experiment similar to the experiments used to determine the linearity of the signal was performed in order to determine the dependence of this background fluorescence observed due to the pump alone. The background fluorescence was found to closely follow a straight line with a slope of 2 on a log–log scale [see Fig. 2(c)]. We consider this evidence that the signal is due to a two photon absorption of ground state HgI_2 producing HgI in the $B\,^2\Sigma$ state. Two photon LIF had been previously reported for HgI_2 by other experimenters.[9,10] In all of the data reported here this back-

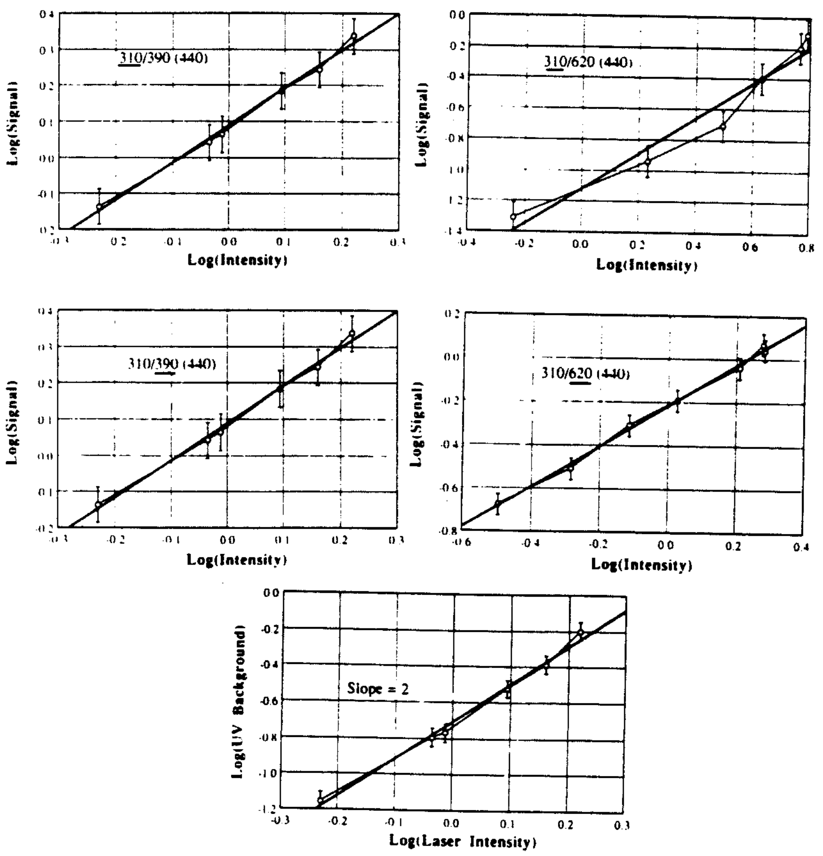


FIG. 2. Power dependence of the LIF signal. The observed LIF is plotted as a function of pump or probe (underlined) intensity on a log–log scale. (a) Pump and probe power dependences for $\lambda_1 = 310$ nm and $\lambda_2 = 390$ nm. (b) Pump and probe power dependences for $\lambda_1 = 310$ nm and $\lambda_2 = 620$ nm. (c) Pump alone, $\lambda_1 = 310$ nm, power dependence of LIF. The heavy lines shown are linear least squares best fits to the data. In (a) and (b) the slopes of the lines are near unity. In (c) the slope of the line is ~2.

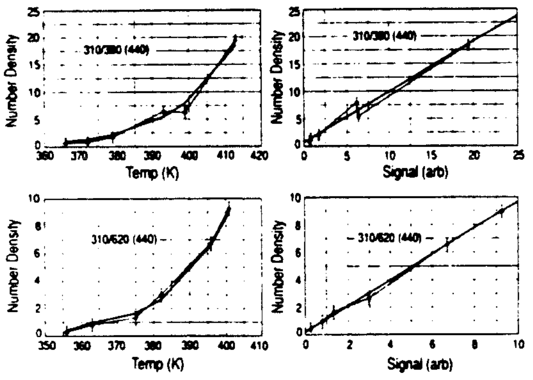


FIG. 3. LIF dependence on number density. The LIF signal and calculated number density are plotted vs temperature along with a plot of LIF signal versus number density. (a) Signals for $\lambda_1 = 310$ nm, $\lambda_2 = 390$ nm and $\lambda_1 = 310$ nm, $\lambda_2 = 620$ nm dependence on temperature (circles) and calculated number density dependence on temperature (heavy line). (b) Calculated number density vs detected signal (circles) for $\lambda_1 = 310$ nm, $\lambda_2 = 390$ nm and $\lambda_1 = 310$ nm, $\lambda_2 = 620$ nm. The heavy lines shown are linear least squares best fits to the data. The slopes are near unity.

ground was always kept to less than 10% of the total signal. The probe beam at 390 nm gave no such fluorescence background. The resulting LIF was dispersed in a monochromator and detection of the signal at wavelengths different than 390 nm assured no scattered light was collected. Probing with 620 nm light gave a very small amount of background fluorescence when allowed in the experimental cell with no attenuation. This background fluorescence was due to a multiphoton excitation. In all experiments this type of background fluorescence was kept to less than 1% of the total signal. It should be noted that the background level due to multiphoton fluorescence was time-delay independent and was easily reduced to a minimum by attenuation of the beams.

The absence of scattered light allowed us to position the gate of the boxcar integrator at time zero and to collect for approximately the width of two fluorescence lifetimes. Several gate widths and positions were tested. In all cases the FTS transient remained unaffected but signal levels and signal-to-noise ratios did change.

The polarization of the pump and probe beams were found to affect the transients but only in a manner predicted by alignment theory.[8(b),15] Good polarization extinction was insured for either parallel or perpendicular pump and probe beams before the nonlinear crystals and after the light had been converted. The effect of parallel vs perpendicular probing has been analyzed in detail.[8] All of the results presented here have been acquired with parallel polarization of both pump and probe beams except where specified.

Finally, the mercury(II) iodide used in this study (Aldrich) had a certified purity of 99.999% and was used directly without further purification. The sealed cell was extensively cleaned with dilute HF solution, rinsed thoroughly with distilled water and then flame cleaned before the introduction of the HgI_2. Following evacuation it was outgassed for several hours and then sealed. The *N,N*-diethylaniline (Aldrich, 98%) was fraction distilled until it became colorless and then was outgassed before every experiment.

III. RESULTS

As mentioned earlier, the HgI_2 reaction provides an extra degree of freedom, namely the participation of the two Hg–I bonds. The unbroken HgI bond may acquire substantial vibrational excitation which is manifested in the fluorescence spectra of the HgI product. Detection of the product LIF at different wavelengths is a new feature for FTS experiments and for its analysis we use a condensed notation. For the pump and probe wavelengths we use the conventional symbols λ_1 and λ_2, respectively. The detection wavelength we call λ_{det}. An experiment resulting from $\lambda_1 = 310$ nm, $\lambda_2 = 390$ nm, and $\lambda_{det} = 440$ nm will be described here as a 310/390 (440) experiment. Using this notation we present our results.

A. 310/390 results

As in an earlier communication,[8(a)] detection at 440 nm yielded the transient shown in Fig. 4(a). The signal consists of a very fast rise followed by a much slower decay to an asymptotic value of about 60% of the peak intensity. In addition to this behavior, oscillations are seen which last for many periods. The first period of 220 fs is different from the longer time oscillations which have a period of ~300 fs. Another transient reported previously was taken at 360 nm and is shown in Fig. 4(b). This transient also has a very fast rise, but then decays to a much smaller plateau value than

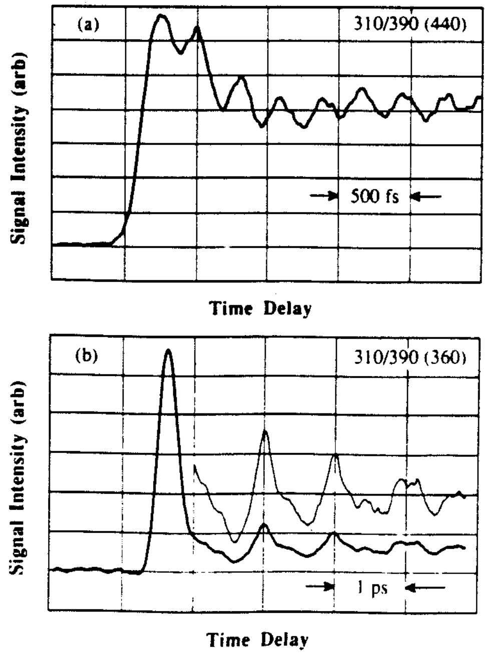


FIG. 4. Typical FTS transients for $\lambda_1 = 310$ nm and $\lambda_2 = 390$ nm. The detection wavelengths used are (a) $\lambda_{det} = 440$ nm and (b) $\lambda_{det} = 360$ nm. The thin line is $\times 3$ expansion of the transient. Note the differences in time scales.

the 440 nm transient, ~10% of the peak value. The oscillatory behavior at this wavelength is quite different than the 440 nm results. The difference between the first and second oscillation is 1.3 ps with the beam frequency then becoming regular at ~1 ps.

To identify the spectral changes we proceeded to disperse the fluorescence arising from the first peak of the transient (near time zero) and then we dispersed the fluorescence from the second oscillation 220 fs later. The two transient spectra are presented in Fig. 5 along with a difference spectrum. The spectrum taken at the first peak has a much broader feature at 440 nm than the more structured spectrum taken 220 fs later suggesting that the 310/390 (440) transient is a combination of (at least) two components. The largest difference between the transient spectra was found at 427.5 and 432.5 nm. Transients taken at these wavelengths are shown in Fig. 6. It should be noted that the slit width was narrowed to 0.5 mm at these wavelengths resulting in a monochromator bandwidth of 1.2 nm. This was as small a bandwidth as possible without cutting the signal-to-noise ratio to unacceptable values. Since these two wavelengths are only 5 nm apart, the narrow bandwidths minimized the spectral overlap of these transients. For all the other experiments mentioned in the results section the slit width had essentially no effect on the transient shape, so the experiments were run with slit widths of 2.0 mm (spectral resolution of 5.0 nm).

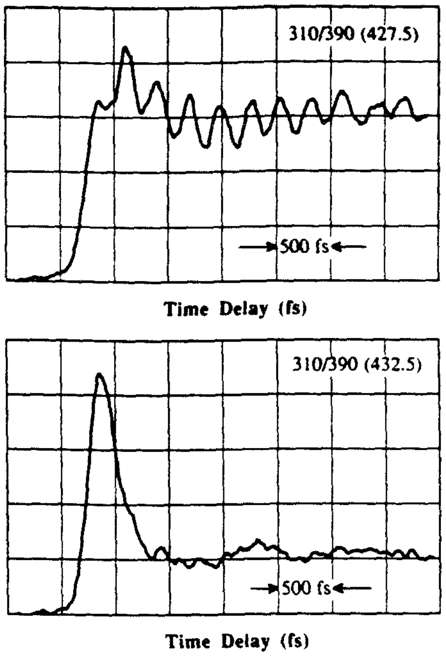


FIG. 6. FTS transients taken for $\lambda_1 = 310$ nm and $\lambda_2 = 390$ nm. The detection wavelengths used are (a) $\lambda_{det} = 427.5$ nm and (b) $\lambda_{det} = 432.5$ nm.

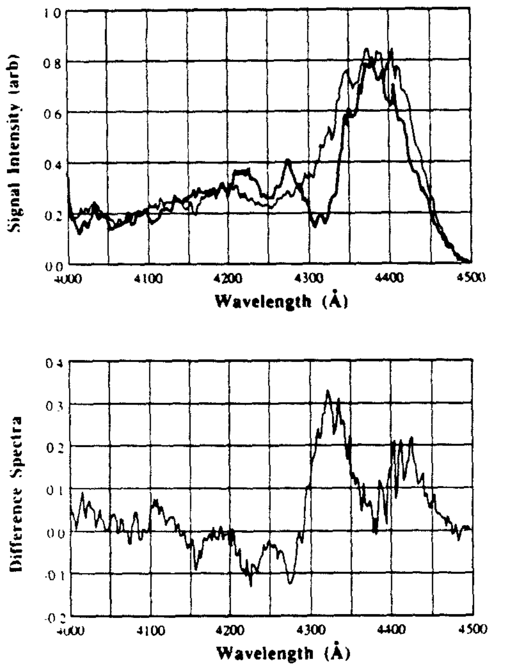


FIG. 5. (Top) Dispersed LIF spectra for $\lambda_1 = 310$ nm and $\lambda_2 = 390$ nm. The light line is for a delay time between pump and probe pulses corresponding to the first peak [in Fig. 4(a)]. The dark line is for a pump-probe delay corresponding to the second peak [in Fig. 4(a)]. (Bottom) The difference spectrum (light minus dark).

The transient taken at 427.5 nm is somewhat similar to the 440 nm transient described above. The major difference is the disappearance of the first oscillation. It now appears as a step instead of a peak. The oscillation frequency after the step is the ~300 fs, the same as reported for the 440 nm transient. In addition, the signal only decays to ~80% of the maximum value. The transient taken at 432.5 nm is almost identical to the 360 nm data. The only difference is the plateau level of around ~20% of peak value is slightly higher than for the 360 nm case. The oscillation pattern is the same, with the depth of modulation being similar.

Transients were recorded for every 5 nm interval from 350 to 450 nm. For all of the other wavelengths examined the behavior was similar to the behavior described above. For the detection wavelengths from 410 to 425 nm and longer than 435 nm the transients were almost identical to those reported at 440 nm. For wavelengths shorter than 370 nm the transients were very similar to the 360 nm transient. For wavelengths close to 390 nm there was a problem with scattered probe light, making it very difficult to record reliable transients.

B. 310/620 results

In this section results are given for $\lambda_1 = 310$ nm and for $\lambda_2 = 620$ nm. In the language of FTS, this corresponds to off-resonance probing, thus $\lambda_2 = \lambda_2^* = 620$ nm. Unlike the 310/390 nm data, the transients taken at these wavelengths all had the same general behavior. Shown in Fig. 7 are the results for three detection wavelengths: 440, 390, and 360 nm. Previously, the 440 nm and 390 nm data have been pub-

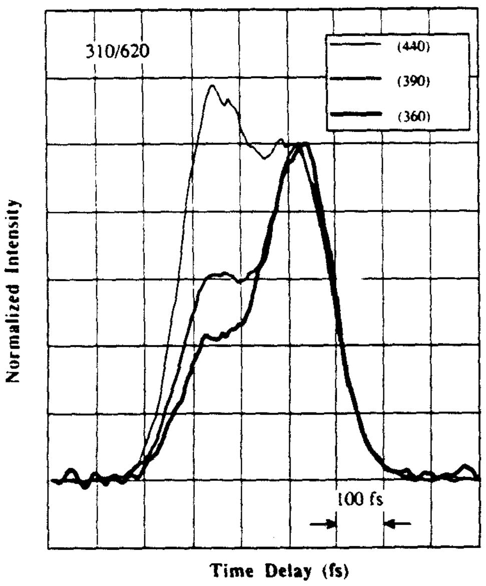


FIG. 7. FTS transients taken for $\lambda_1 = 310$ nm and $\lambda_2 = 620$ nm. The second peak intensity has been scaled to an intensity of 1.0 so comparisons can be made. The detection wavelengths for each transient are indicated in the legend.

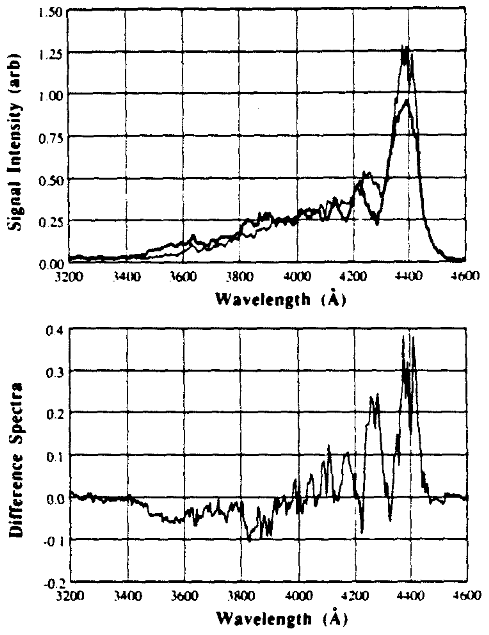


FIG. 8. (Top) Dispersed LIF spectra for $\lambda_1 = 310$ nm and $\lambda_2 = 620$ nm. The light line is for a delay time between pump and probe pulses corresponding to the first peak (in Fig. 7). The dark line is for the pump–probe delay corresponding to the second peak (in Fig. 7). (Bottom) The difference spectrum (light minus dark).

lished[8(a)] for perpendicular polarization of the pump and probe beams. All the transients consist of two peaks, i.e., a single oscillation. The only difference in these transients is the relative height of these peaks with respect to each other. Other features are an instrument response limited rise time and a decay to zero signal at times greater than 600 fs; this is a completely off-resonance transient. As the detection wavelength moves to the blue, the amplitude of the second peak grows with respect to the first peak (or the first peak amplitude decays with respect to the second).

Spectra were taken from 320 to 460 nm at probe time delays corresponding to the first peak and the second peak. These spectra are shown in Fig. 8. Again the spectra are very different. The spectra were subtracted and their difference spectrum is structured. Differences in the spectra will be discussed in the following sections.

IV. DISCUSSION

A. Preliminaries

The dissociation of HgI_2 occurs following excitation of the first long wavelength absorption band peaking at 265 nm and extending to 350 nm.[9,10] Quantum yield of iodine in both spin–orbit states has been determined by Leone's group.[10(a)] Production of $I^*(^2P_{1/2})$ was found to be favored for excitation at wavelengths shorter than 295 nm while at longer wavelengths the ratio of $I(^2P_{3/2})$ dominates. At 310 nm, the wavelength of interest for this study, one can estimate the production of $I^*(^2P_{1/2})$ to be $\sim 25\%$.[10(a)] Formation of the two spin–orbit states of iodine is believed to arise from separate potential energy surfaces of HgI_2.

The energetics of the reaction for HgI_2 vapor at 150 °C have been estimated as follows: Firstly, the thermal energy is estimated to be $5\,kT \simeq 1500$ cm^{-1} (for rotation and vibration of a linear triatomic). Secondly, the dissociation energy for the reaction $HgI_2 \rightarrow$ ground state HgI and I is $\simeq 21\,000$ cm^{-1} (the I–HgI is stable by -1.45 eV with respect to Hg and I_2 and the endoergicity of the reaction $Hg + I_2 \rightarrow HgI + I$ is 1.15 eV resulting in 2.6 eV for D_0 as determined by Wilcomb *et al.*[12]). Thirdly, the photon energy of the pump pulse at 310 nm is 32 300 cm^{-1}, adding the thermal energy gives a total of 33 800 cm^{-1}. Subtracting the dissociation energy from the total energy in the excited molecule yields $\sim 12\,800$ cm^{-1} of excess energy available for the recoil of the fragments and the vibration and rotation of the HgI fragment. If the rotational energy is $\simeq 300$ cm^{-1} (assuming a Boltzmann distribution at 150 °C) it turns out that 12 500 cm^{-1} of energy is available for translation and vibration of the products. When the photodissociation involves the production of $I^*(^2P_{1/2})$ the spin–orbit coupling energy of 7600 cm^{-1} must be subtracted leaving this channel with 4900 cm^{-1} of available excess energy. Other useful constants are given in Table I, including the vibrational frequencies for HgI_2 and HgI.

In the following sections we discuss: the effect of FTS probing at different wavelengths, the discrimination of reaction trajectories accessed by different wavelength detection and how the FTS transient obtained are utilized to map the PES of the reaction.

B. FTS of the reaction

In light of past FTS studies on ICN[4] and alkali-halides[6,7] we first present an interpretation of the experiments

TABLE I. Spectroscopic constants for HgI_2 and HgI.

	HgI_2 (ground state)		ν_0 [a]	Period
Q	Symmetric stretch		155 cm^{-1}	215 fs
Q_a	Antisymmetric stretch		237 cm^{-1}	141 fs
q	Bend		33 cm^{-1}	1000 fs
D	Dissociation energy		21 000 cm^{-1}	
HgI ($X\,^2\Sigma$)[c]	ω_e [b]	125 cm^{-1}		period 267 fs
	D	2800 cm^{-1}	(dissociation energy)	
	β	7.09	($\beta^2 = \omega_e^2/4B_e$)	
	r_e	2.80 Å		
HgI ($B\,^2\Sigma$)[c]	ω_e [b]	110 cm^{-1}		period 303 fs
	D	18 850 cm^{-1}		
	β	2.86		
	r_e	3.30 Å		

HgI ($B-X$) Transition dipole $\mu = 4\exp[-(r-3.39)^2/0.9^2]$ Debye

[a] Values taken from Lowenschuss, Ron, and Schnepp, J. Chem. Phys. **50**, 2502 (1969).
[b] Values taken from Ref. 9(b).
[c] Values for D, β, and r_e have been chosen to give the best overall representation of the ground state potential energy function and the best representation of the accessed vibrational energy levels of the excited state.

treating the dissociation as occurring on a one-dimensional PES (see Fig. 9). The PES's drawn in Figs. 9 and 10 have been scaled with available spectroscopic[9-12] and thermodynamic[16] data and are helpful in presenting a first order understanding of the FTS transients obtained. The full two-dimensional interpretation of the results is left for the next section. Figure 9 shows the schematic for the pump–probe experiment. The pump pulse accesses both the I and I* PES (directly or indirectly through crossing) depositing a wave packet at time zero of the reaction. The probe wavelength, drawn to scale, interrogates the wave packets as they make their way to final products. In Fig. 10 we show all of the states accessible in our experiment, along with the HgI absorption shown on the left axis. Figure 11 is a schematic explaining how we detect different vibrational levels with fixed pump and probe wavelengths. Using the proper Franck–Condon factors and vibrational distributions (see the next section), we find that detection at 360 nm accesses high v'' in the HgI ground state and similarly detection at 427.5 nm accesses low v'' in that same state.

For all the experiments, the pump wavelength λ_1 was fixed at 310 nm while two probe wavelengths were used. The probe wavelength was either on resonance λ_2^∞ for 390 nm or off-resonance λ_2^* for 620 nm (see Fig. 9). The distinction between on and off resonance is made here by the fact that for on-resonance probing, at 390 nm, the free HgI fragment absorbs the probe wavelength giving rise to an asymptotic

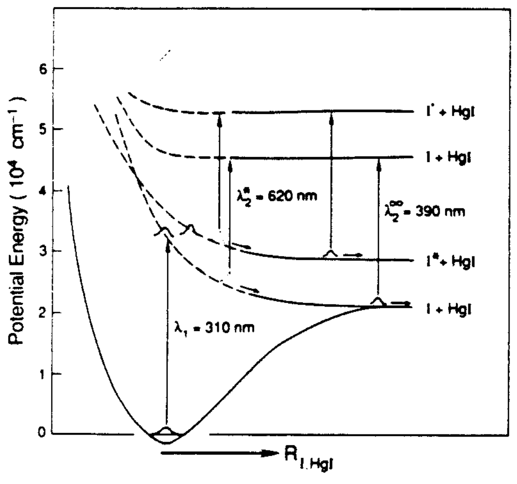


FIG. 9. Schematic of the FTS experiment with the pump and probe energies drawn to scale. At time zero the pump pulse deposits a wave packet in the I and I* channels. The probe pulses interrogate the motion of the wave packets at early times (620 nm) in the transition-state region or at later times (390 nm) as they become free fragments.

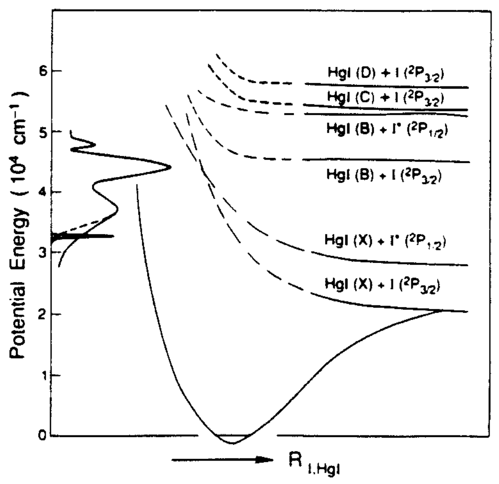

Potential Energy (10^4 cm^{-1})
6
5
4
3
2
1
0
HgI (D) + I ($^2P_{3/2}$)
HgI (C) + I ($^2P_{3/2}$)
HgI (B) + I* ($^2P_{1/2}$)
HgI (B) + I ($^2P_{3/2}$)
HgI (X) + I* ($^2P_{1/2}$)
HgI (X) + I ($^2P_{3/2}$)
$R_{I,HgI}$

FIG. 10. Schematic of the one-dimensional PES of the HgI_2 dissociation. The absorption spectrum of HgI_2 [Ref. 10(a)] and the pump energy used are sketched on the left.

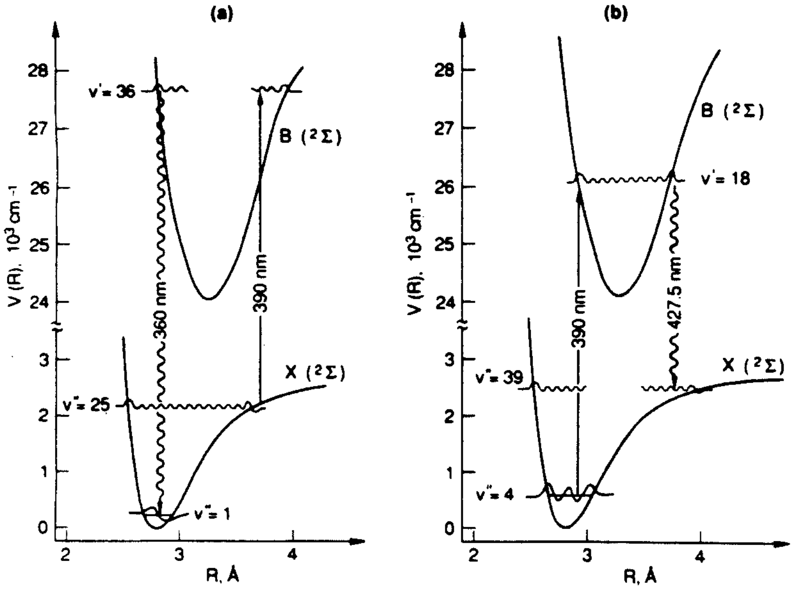


FIG. 11. Schematic of the HgI PES showing the effect of probing with 390 nm two vibrational distributions. (a) Probing a high vibrational level $v'' = 25$ leads to fluorescence at 360 nm. (b) Probing a low vibrational level $v'' = 4$ leads to fluorescence at 427.5 nm.

level in the transients where the coherent oscillations are present. In the off-resonance case, there is no transition of the free HgI which is resonant with the 620 nm probe. It is evident in the experimental transient that for 620 nm excitation the asymptote is at the same level as the base line, implying that there is no signal at longer times and that only fragments in the transition-state region absorb. Off-resonance signal is usually characterized by a rise from zero signal and decay to zero signal at short times. These rise and decay times can be related to the lifetime of the transition-state of the reaction for a given spectral window determined by the probe spectrum and convolved by the temporal duration of the pulses.[5]

Probing at $\lambda_2^\infty = 390$ nm we observe several types of transients which appear to be combinations of two distinct behaviors as seen in the data for 310/390(427.5) and 310/390(432.5). The transient obtained for 310/390(427.5) basically reaches a maximum and remains at the maximum level of signal for all times (neglecting the oscillations). The transient for 310/390(432.5) after the first peak near time zero shows the growth of an asymptotic level which shows long period oscillations lasting for long times.

For the analysis it is important to determine whether these two transients are spectrally pure. Subtracting 15% of the (427.5) transient from the (432.5) transient we obtain a transient that is very similar to the data taken at 360 nm [see Fig. 12(a)]. Thus, it appears that these transients are not entirely separated spectrally. Subtracting 67% of the (432.5) transient from the (427.5) data we obtain the tran-

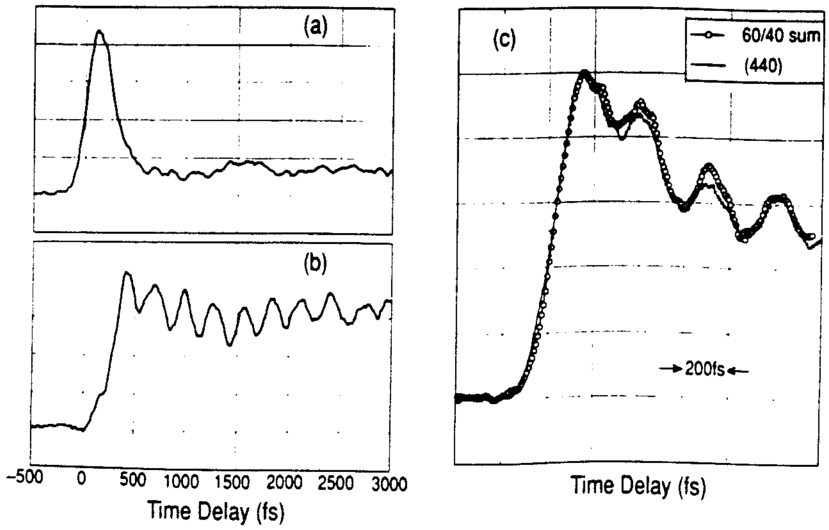


FIG. 12. (a) Transient resulting from subtracting 15% of the unnormalized 310/390 (427.5) transient (Fig. 6 top) from the unnormalized 310/390 (432.5) transient (Fig. 6 bottom). (b) Transient resulting from subtracting 67% of the unnormalized 310/390 (432.5) transient (Fig. 6 bottom) from the unnormalized 310/390 (427.5) transient (Fig. 6 top). Notice that ~300 fs delay between transient (a) and (b) indicating "on- and off-resonance-like" behavior. (c) Transient (circles) resulting from addition of 60% of the unnormalized 310/390 (427.5) transient with 40% of the unnormalized 310/390 (432.5) transient. The transient has been scaled to show the similarities with the 310/390 (440) transient published in Ref. 8(a) (black squares).

sient shown in Fig. 12(b). There is clearly a delay from time zero before the signal begins to rise (see below). So the data taken detecting at (427.5) and (432.5) are made up of the two pure transients shown in Figs. 12(a) and 12(b). As further proof in Fig. 12(c) we show that the previously published 310/390(440) transient can be understood as a combination of the (427.5) and (432.5) data. This is clear indication that all of the HgI_2 experiments performed at 310/390 give rise to two distinct behaviors: a transient similar to the 310/390(360) data and the transient shown in Fig. 12(b).

The analysis of these FTS on-resonance transients yields the "clocking" time of the reaction.[4] Briefly, the clocking experiment can be described as follows. During the dissociation the fragments are in the process of becoming free particles. The internuclear distance between the fragments determines the extent of the perturbation that one fragment has on the other. The clocking experiments determines at which point in time (and distance) the products are spectroscopically distinguishable as free particles. Fully on-resonance FTS transients have a delay time ($\tau_{1/2}$) that is associated with the time it takes the free fragment absorption to "turn on." This can be clearly seen in Fig. 12(b), where a $\tau_{1/2}$ of ~300 fs is found.

The transients obtained with $\lambda_2^* = 390$ nm can be understood in terms of two reaction channels producing two types of behaviors. As mentioned in the results section the period separation gives the vibrational energy content of the HgI fragment and we can therefore determine the amount of recoil energy. From the transient in Fig. 12(b) we consider the frequency of the oscillations (~300 fs) and we match it to the corresponding vibrational spacing of the HgI in the ground state (for this case $v'' = 7 \pm 1$). Using a similar calculation for the transient in Fig. 12(a) we obtain $v'' = 29 \pm 2$.

From these numbers it is possible to determine the amount of vibrational excitation in the dissociating fragments. Remembering, that energetically it is possible to produce iodine in its ground state ($^2P_{3/2}$) or the spin–orbit excited state ($^2P_{1/2}$), gives four possibilities for the total amount of energy for recoil [11 800 or 10 000 cm^{-1} for the $I(^2P_{3/2})$ and 4200 or 2400 cm^{-1} for $I^*(^2P_{1/2})$]. These energies can now be converted to the terminal velocity in the center of the mass of the fragments. For now, assume that the I channel leads to HgI with high vibrational excitation, i.e., 10 000 cm^{-1} are available for translation, while the I* channel leads to low vibrational excitation, i.e., 4200 cm^{-1} go into translation. Thus, the terminal velocity at which I recoils after dissociation is 1.6×10^{-2} Å/fs while the I* terminal velocity is calculated to be 1.0×10^{-2} Å/fs. Taking the I and I* PES to be simple one-dimensional exponentials with a typical length parameter ($L = 0.6$ Å), we can estimate the time of arrival of free HgI to be 220 and 300 fs for the I and I* channels,respectively. These calculations give a good estimate for the dissociation dynamics but the PES for both channels has been oversimplified and more accurate calculations require full consideration of other degrees of freedom as discussed later.

The (360) transients show a fast rise (that is very close to time zero) and then a fast decay. In addition, the asymptote level is a small percentage of the peak signal. There are two possible explanations for this large signal intensity at early times: the most favorable transition dipole moment is reached early in time near the transition-state region and/or the transition-state complex probed at early times undergoes complete dissociation to Hg + I + I. From the diagram in Fig. 10 it is clear that higher states are accessible with 390 nm probing at very short internuclear distances. We have recorded a transient detecting at 290 nm, probably due to fluorescence from the C or D states, which are accessed very near the time zero of the reaction by the 390 probe laser.

The off-resonance signals obtained with $\lambda_2^* = 620$ nm probe the transition-state region. At every detection wavelength the form of the transients is the same, consisting of two peaks separated by ~220 fs. The spectra taken at each of these peaks are different (see Fig. 8) indicating that these transients may come from two different channels (similar to the explanation for the 310/390 experiments). This is easily understood; since the channels have different energetics, the time it takes for each to arrive in the probing region could certainly be different (the time between the two peaks). If we consider that the two peaks are independent signals, then the width of the peaks can give an estimate of the life time of the transition-state accessed at this wavelength. This lifetime is on the order of 50 fs, implying that the probing is done on a steep (highly repulsive) part of the potential.

The interpretation of the HgI_2 transients we have obtained in terms of one-dimensional FTS theory is rather limited. In order to fully appreciate the molecular dynamics for this triatomic molecule one more coordinate is needed. It is the motion along the two Hg–I coordinates that determines the behavior of the transients. Classical, semiclassical, and quantum mechanical simulations have been performed (see below) in order to determine the trajectories responsible for the transients obtained experimentally. The results of these simulations and further refinement in our understanding of the reaction follows.

C. Two-dimensional FTS and the potential surface

We now turn to a more quantitative, yet still very simple analysis of the FTS experiments in terms of classical trajectories on a two-dimensional PES for the I–Hg–I system. In part 1, we review our choice of the PES, and the results of the trajectory calculations. In part 2, the quantum mechanical calculation of the HgI fluorescence and a semiquantitative comparison with the transients shown earlier will be presented.

1. PES and classical trajectory calculations

The potential energy function of HgI has been studied extensively by emission spectroscopy,[9(b),10(b)] and the asymptotic behavior of the full PES, insofar as it can be represented by Morse potentials, is well known. On the other hand, the nature of the $^1\Sigma_g^+$ HgI_2 surface giving rise to the HgI (X) + I and HgI(X) + I* channels accessed by the pump beam, and of the $^1\Sigma_u^+$ surface giving rise to the HgI(B) + I and HgI(B) + I* states accessed by the probe beam, is largely unknown. Relativistic *ab initio* calculations on $HgCl_2$ by Wadt[17] indicate that these states will be bent by 80°–100°. However, the vertical pump transition from the ground state will most likely access the upper state in a linear configuration, given that the bending mode of HgI_2 is only thermally excited. Furthermore, the coupling of the bending motion with fragment rotation and translation occurs on a time scale of picoseconds and plays a lesser role in the bond breaking process near the transition state. The rotational dephasing of HgI_2 has been discussed in detail in Ref. 8(b); we will concentrate on a two-dimensional approach involving the asymmetric and symmetric stretching coordinates of HgI_2 and translation and stretching vibration of the fragment instead. We thus confine ourselves to an understanding of the motions within the first 1–2 ps, after which the breaking of the bond is complete, with the understanding that any PES described below represent appropriate rotational and bending vibrational averages.

Since reliable knowledge about the PES is limited, an empirical surface, such as a LEPS potential, must be used.[18] We have chosen a more flexible damped Morse surface, which can be used in quantitative fitting of the FTS spectra, when combined with a fully quantum mechanical treatment. Its general form is

$$V(r_1 r_2) = D(1 - \exp\{-\beta[e^{-\gamma r_1}(r_1 - r_e)/r_e + e^{-\gamma r_2}(r_2 - r_e)/r_e]\})^2 - D, \qquad (4)$$

where r_1 and r_2 are the two Hg–I bond lengths; D, β, and r_e are the dissociation energy, Morse coefficient, and equilibrium bond length along the reaction coordinate, and may depend on the deviation of the triatomic arrangement from a symmetric configuration; for the present purpose, they were represented as

$$f = f_0 + f_1 \exp[-(r_1 - r_2)^2/\sigma^2] \qquad (5)$$

with $\beta_0 = 7.09$, $\beta_1 = 0.5$, $D_0 = 2800\ \text{cm}^{-1}$, $D_1 = -1000\ \text{cm}^{-1}$, $r_{e0} = 2.8$ Å, $r_{e1} = 0.4$ Å, $\sigma_{\beta,r} = 1.5$ Å and $\sigma_D = 1$ Å. This yields a good approximation to the potential function of HgI in the separated fragments limit, with a dissociation energy (in the symmetric stretch direction) of $1800\ \text{cm}^{-1}$ at the saddle point, and a symmetric stretching frequency at the transition state slightly higher than the HgI stretching frequency. The tightness of the transition state is determined by β_1, which gives a very loose transition state when negative, and an increasingly tight transition state when positive. γ determines how fast the PES approaches an asymptotic I + HgI or IHg + I channel, and was chosen to be 1.5 Å. The resulting PES is shown in Fig. 13; with the above choice of parameters, the asymptotic region is well represented, and the transition region resembles a LEPS potential derived from the positive and negative contributions to the Morse potential of HgI. Of all PES tested, the above agreed best with the experimental measurements.

Before attempting a calculation, another simplifying assumption was made. Since the spin–orbit coupling is about $7600\ \text{cm}^{-1}$, while the dissociation energy of HgI is only about $3000\ \text{cm}^{-1}$, a crossing of the surfaces leading to the I and I* channels at internuclear separations larger than the ground state geometry is unlikely to yield the observed results, although it cannot be excluded as a possibility. Rather than postulating excitation by the pump beam to a single (I*) surface, followed by Landau–Zener crossing, we assume the picture of two nearly parallel surfaces leading to the I and I* channels, as shown schematically in Fig. 10. Furthermore, the two PES will be taken to differ only by the spin–orbit splitting in iodine, i.e., they are vertically shifted versions of the same functional form Eq. (4). That this assumption is reasonable can be seen if the I/I* branching ratio measured by Hofmann and Leone[10(b)] is interpreted in terms of two independent potential surfaces: I* formation is resonant at 265 nm, while I formation is resonant at 310 nm at the ground state equilibrium geometry. This $5500\ \text{cm}^{-1}$ difference is still fairly close to the iodine spin–orbit splitting, and any surface crossings can reasonably be assumed to take place at shorter bond distances. The intensity ratio of the I/I* channels will then be determined by three effects: The Franck–Condon overlap with the ground state wave function(s), which favors the I channel; the possible decay of HgI_2 into I + Hg + I, which becomes energetically accessible at higher excess energies ($\lambda_1 < 355$ nm) and favors the I* channel; finally, the transition dipole matrix element, which probably favors the I* channel.[10(a)] The second contribution can be easily estimated from classical trajectory calculations, while the others require more detailed knowledge of the PES and wave functions. In general, we will assume the measured branching ratio of about 3.7:1 I:I* at $\lambda_1 = 310$ nm *ad hoc*. With these simplifications, the I and I* channels differ only by the excitation energy above the as-

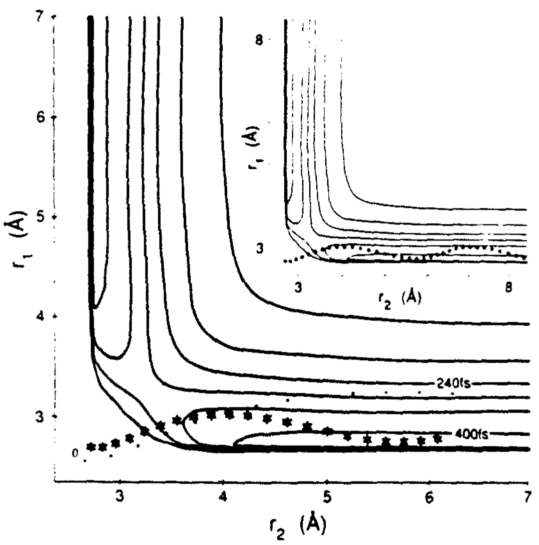


FIG. 13. Contour plot of the potential surface from Eq. (4). The tightness of the transition scale can be adjusted by varying β_1, which was found to be the most important parameter in reproducing the experimental results. The averaged trajectories for I* (*) and I (·) are shown; they originate at the turning point corresponding to the particular energy of excitation. Note that no oscillations are executed near the transition state, as the reaction time is comparable to the vibrational period of reactant and products, and that the I* channel corresponds to faster, smaller amplitude vibrations. The insert shows an expanded view where two full oscillations are shown.

ymptote, and the position of the classical turning point at which the wave packet originates.

We use the standard Hamiltonian

$$H = \frac{1}{\mathscr{M}}(p_1^2 + p_2^2) + \frac{1}{\mathscr{M}_{12}} p_1 p_2 + V(r_1 r_2), \qquad (6)$$

where the p_i are the momenta canonically conjugate to the HgI bond distances in the three-particle center of mass coordinate system, and the $\mathscr{M}$'s are the appropriate effective masses. The calculation was performed at excitation energies of 8950 (I) and1350 cm^{-1} (I^*) relative to the HgI dissociation limit, by importance sampling a Gaussian distribution of starting points with a width determined by the excited state potential gradient, laser bandwidth, and approximate width of the ground state probability density. Momenta were chosen with random orientations but satisfying energy conservation. Integration of the equations of motion was carried out using Adam's predictor corrector method,[19] until the trajectories had become asymptotic ($r_i > 6$ Å). In view of the strong assumptions already made concerning the potential functions, this very simple treatment of the initial conditions is justified, particularly since the excitation energies used in the present study access only unbound states of the I–Hg–I system. This fact, combined with excitation of sufficient bandwidth, leads to the formation of initial superposition states localized near the inner turning point of the transition state. As a check of stability, several calculations were also carried out with a LEPS potential, different parameters in Eq. (4), and different choices of excess energy and width of the initial position distribution. The results were qualitatively the same, and we have chosen the above parametrization and set of assumptions because it yielded the best overall agreement with experiment.

The most important results of two hundred simulations for each channel are summarized in Table II. Averaged trajectories for the I and I^* channels are shown in Fig. 13. They were obtained by summing trajectories going into the same final channel. The I channel, with its higher excess energy, moves to the products faster (<240 fs), and has a lower vibrational frequency (HgI anharmonicity) and larger vibrational amplitude. The I^* channel moves more slowly (<400 fs reaction time), and has less internal energy, corresponding to more rapid, smaller amplitude vibrations. Since the two channels in this simple model differ only by their excitation energy and trajectory origins, this "equipartition" of higher available energy leading to higher internal and translational excitation is to be expected (within a single channel, higher vibrational energy of course entails lower translational energy, and *vice versa*).

In either case, the reaction time is sufficiently short that no oscillations are executed near the transition-state region. In the TS region, where probe absorption at 620 nm occurs, the I^* channel wave packet lags the I channel by about 80 fs. The I channel also has a smaller acceptance angle for decay into I + HgI; 60% of the trajectories at 8950 cm^{-1} excess energy above the HgI dissociation limit actually dissociate into I + Hg + I; while full dissociation is also energetically allowed in the I^* channel, almost none occurs. In the early time motion near the saddle point, product translation and vibration are strongly coupled as the normal coordinates are intermediate between symmetric/asymmetric stretching and HgI vibration/translation; this could account for the shifts in vibrational period observed for the first two peaks in Fig. 4(b), although it is also possible that the first peak has a large contribution from total dissociation.

The vibrational distributions of the HgI product in the two channels were found to be peaked at $v = 10$ in the I^* channel and $v = 26$ in the I channel. Figure 14 shows typical distributions obtained from 100 nondissociative trajectories for each channel. None of the surfaces tried here produced bimodal distributions which could explain the observed results by a single channel model (other than Landau–Zener crossing).

2. HgI fluorescence spectra and the observed FTS transients

In order to compare the above results quantitatively with experiment, the HgI product fluorescence resulting at various probe times and frequencies from the two channels must be known. We will concentrate on the 390 nm probe experiment, since they correspond to separated HgI fragments, for which the spectroscopy is known. The potential parameters for the HgI X and B states in Table I represent an average of the best available literature values. The Morse

TABLE II. Trajectory calculations for the I and I^* channel, using the PES in Fig. 13.

	I^* channel	I channel
Excess energy[a]	1350 cm^{-1}	8950 cm^{-1}
Reaction time[b]	400 fs	240 fs
Full dissociation[c]	4%	60%
Average HgI v''[d]	13 ± 8	22 ± 8

[a] Energy above the I–Hg–I total dissociation limit distributed in the two collinear degrees of freedom.
[b] Here simply taken as the time required for trajectories to reach the asymptotic region of the PES at r larger than 7 Å; the temporal dependence of the position of the wave packet center is given in Fig. 13.
[c] To Hg + 2I or Hg + I + I^*.
[d] Values are average (not peak) and standard deviation.

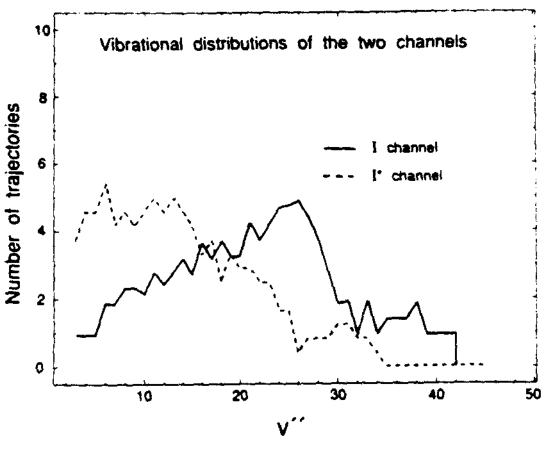


FIG. 14. HgI product vibrational distributions for the I channel (solid) and I^* channel (dashed).

representations and transition dipole are sufficiently accurate such as not to introduce any noticeable spectral changes for a laser bandwidth in excess of 100 cm^{-1}.

The spectra are calculated by numerical integration of the vibrational Schrödinger equation in the following steps.[20] All X manifold bound states and vibrational wave functions are calculated. All B manifold states accessible by a 390 nm probe are calculated; in order to include emission into the continuum, at X manifold continuum wave functions accessible from the B manifold at several frequencies in the 420–460 nm range are evaluated. This results in a full set of Franck–Condon factors, which were found to be in good agreement with those previously reported.[9(b),10(b)] A lower state population distribution (Fig. 14) is assumed, and the distribution of B states from absorption of 390 nm radiation of the appropriate bandwidth is calculated. For each B state, the fractional parentage of X vibrational states is stored. The emission spectrum back to the ground electronic state is then evaluated, and for each transition, the original parent v'' distribution before pumping is known. The calculation was also performed with a $J = 100$ rotational term in the HgI potential function, and found to yield very similar results. Rotation was therefore again neglected, with the same understanding that the effective PES for the I–Hg–I system is rotationally averaged and cannot explain "slow" modulations of the data resulting from rotational anisotropy (see below). Finally, the experimental branching ratio of the I* and I channels was included to weight the individual contributions of the two channels to the overall fluorescence.

The results for the I* and I channels are shown in Figs. 15 and 16. The contributions of the two channels to the total fluorescence intensity are seen to be very different. The dashed band shows the spread of originally probed (v'') vibrational states as a function of frequency. The general trend is a slow decrease in v'' until continuum emission sets in at ~430 nm, where v'' rises again. For example, if one could somehow look at fluorescence of HgI formed in the I* channel only, detecting at 360 nm would reveal the dynamics of reactions yielding HgI at about $v'' = 22$, whereas detection at 428 nm would reveal the dynamics leading to $v'' = 5$ product. Fortunately, the two channels have widely varying contributions at different frequencies. The ratio of I/I* intensities is shown in Fig. 17, and in terms of it we can understand the 310/390 transients: where the ratio is larger than 1, the I channel dominates, where it is less than 1, the I* channel is more important. We now consider the various emission wavelengths of the 310/390 experiment in turn.

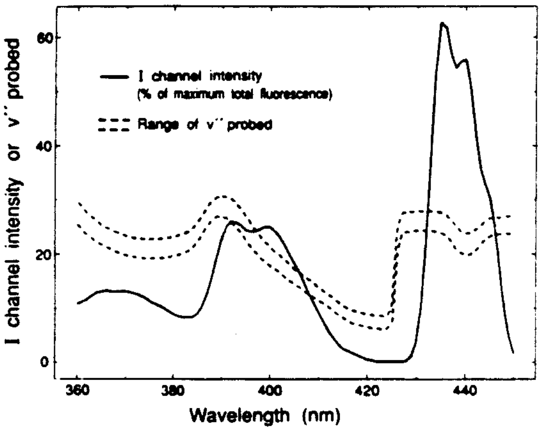


FIG. 16. Same as Fig. 15 for the I channel. The average vibrational energy of probed product HgI is much higher in this case.

At a detection wavelength of 360 nm, the I channel dominates. That transient probes highly excited ($v'' \sim 28$) molecules with a large internal and translational energy. The vibrational period at $v'' = 28$ is about 1 ps, as observed experimentally in the recurrences [Fig. 4(b)]. The very large first peak most likely also has a contribution from the I + Hg + I channel, which makes up 60% of the product

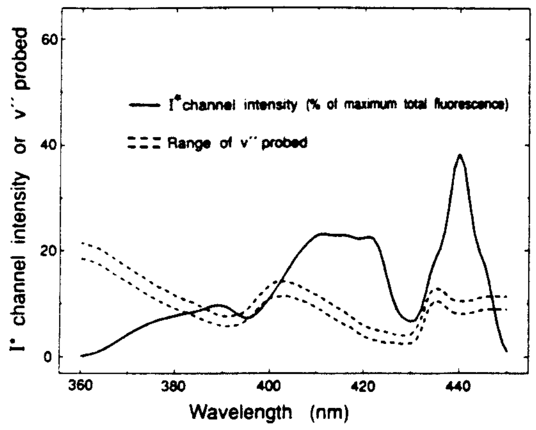


FIG. 15. Contribution on the I* channel to the total fluorescence spectrum, assuming a product vibrational distribution as calculated in the classical trajectory simulations; this includes the I/I* 3.7:1 observed branching ratio. The dashed lines bracket two standard deviations of the vibrational product distribution contributing to a given emission frequency.

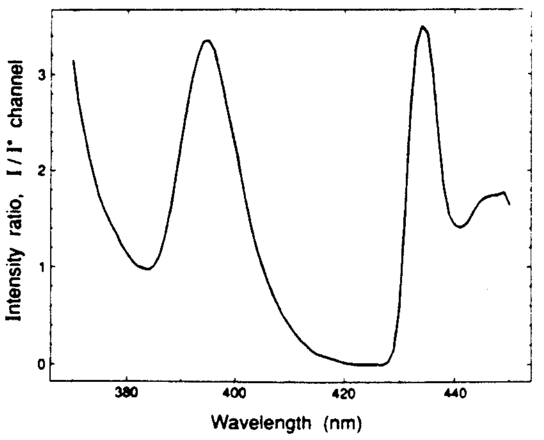


FIG. 17. The contributions of the two reaction channels to the product fluorescence vary as a function of frequency. The 360 and 432.5 nm fluorescence is seen to be due mainly to the I channel, whereas the 427.5 nm fluorescence is almost exclusively due to the I* channel; the 440 nm fluorescence contains contributions from both.

from the lower lying I surface (the I + Hg + I* channel is essentially closed at 310 nm, see Table II). At 427.5 nm, the I* channel is the most important contribution. As seen in the trajectory calculations, the product vibrational distribution is much lower. At 427.5 nm, the emission spectrum probes vibrational states around $v'' = 5 \pm 2$, in good agreement with the 300 fs oscillation period in Fig. 6. Resonance has not yet been completely achieved at the first peak, which is truncated. At 432.5 nm, the continuum emission becomes important, and the I channel is most important again. Hence, the transient again shows a large first peak (perhaps partly due to full dissociation) followed by recurrences corresponding to a distribution of HgI states around $v'' = 28$, as seen in Fig. 6. Finally, at 440 nm, the contribution of the channels is I*:I 40:60, close to the experimentally observed spectrum, which shows rapid oscillations corresponding to lower vibrational excitation in the I* channel, and a larger first peak corresponding to the I channel [Fig. 4(a)]. To a good approximation then, the transients at various frequencies can be constructed by averaging together pure I (large first peak, ~1 ps recurrences) and I* (oscillations near 300 fs) transients with weights and vibrational frequencies appropriate for the detection frequency.

The ratio in Fig. 17 is qualitatively very similar to the difference spectrum in Fig. 5. At 360 and 432.5 nm, the difference is a large positive fraction of the total signal; the first peak in the FTS transient is being probed, which contains more I channel signal, corresponding to a ratio above unity in Fig. 17. At 427.5 nm, the difference becomes a sizeable negative fraction of the total signal: The second peak, mostly due to the I* channel, contributes more, and the ratio in Fig. 17 is correspondingly small. Finally, at 440 nm, the difference is small compared to the total signal: I and I* contribute equally, and the ratio in Fig. 17 is near unity. It should be noted, however, that in the region between 400 and 420 nm, the difference spectrum predicts the I and I* channels to contribute equally, while the ratio in Fig. 17 indicates a progressive change from an I to an I* type transient. The region near 395 nm, where the difference is dramatic, is not easily accessible experimentally. Furthermore, the width and height of the local maximum near 395 nm in Fig. 17 is fairly sensitive to the choice of vibrational distribution, and could lead to a refinement of the potential function with a more complete theoretical treatment.

Dispersed fluorescence probing is thus a very powerful tool in understanding the FTS behavior. Looking at certain well-defined frequencies allows one to probe a specific reaction channel, and even specific vibrational levels in the reaction channel. Although the lack of information on the PES does not allow such a quantitative comparison for the 310/620 experiments, which probe the reaction closer to the transition state, a few salient features emerge by comparing the transients with the classical trajectory calculations and fluorescence simulations. Referring to Fig. 13, the difference in average translational energies can account for a 100 fs lag of the I* channel wave packet with respect to the I channel wave packet at 4–5 Å into the dissociation. Based on this, one would expect two peaks in the FTS transients, the first one about 80 fs from time zero (I), the second another 100 fs later (I*). This corresponds closely to the observations presented in Sec. III B. The actual peak separation is twice as wide; this may reflect a shift in the transition frequency for the probe beam, since the trajectories do not follow the same path, and the excited state PES leading to *B* HgI in the I and I* channels need not be identical in shape.

To summarize, the classical trajectory calculations combined with a quantum mechanical evaluation of the HgI emission spectra, are in good agreement with many of the observed features of the FTS spectroscopy. The two stretching coordinates are the most important on the time scale of the bond breaking, and the two-dimensional model looks like a promising basis for a more extensive treatment of the problem including bending modes and fragment rotation. A full quantum calculation will be reported in a separate publication.[21]

V. CONCLUSIONS

In this paper, we have presented an FTS study of the HgI_2 reaction:

$$\mathrm{IHgI}^* \rightarrow [\mathrm{IHgI}]^{\ddagger *}_{Q_r, Q_s, q} \rightarrow \mathrm{HgI} + \mathrm{I}.$$

The molecular motions on the femtosecond time scale of the dissociation process, which involve two predominant degrees of freedom, were time resolved. Their fingerprint was found in the oscillatory behavior of the transients, which open a direct experimental window on the transition state, as the symmetric and antisymmetric stretching modes of the parent HgI_2 are converted to the translational and single vibrational degree of freedom of the HgI + I products.

Several experimental parameters were varied to study the details of the reaction: The choice of probe wavelength allowed a probing of the dissociating wave packet on early and late time scales. The fluorescence spectrum of HgI, pumped at 390 nm, revealed a richness of detail, with different spectral regions resulting from different channels of the dissociation reaction. This yielded dramatically different vibrational oscillatory patterns in transients at different wavelengths, due to the different vibrational and translational energy distributions in the I and I* channels.

We have also studied, and reported previously,[8(b)] the polarization anisotropy of the spectra, an example of which is shown in Fig. 18.[22] This way both the "scalar" and "vector" properties of the reaction were obtained. From the data presented in this paper we found a dissociation time of ~300 fs, and different vibrational periods (~300 fs and 1 ps) for the nascent HgI product along the two (I and I*) channels. From the data in Fig. 18 we obtained a coherence time of ~1 ps and deduced the angular momentum of the fragment $J_{max} \sim 80$.

The intuitive simplicity of the FTS transients has been put on a more quantitative foundation by comparison with classical trajectory calculations of the dissociation reaction and quantum calculations of the HgI fragment spectroscopy. Even the simplest two surface model, neglecting any possible crossings, yields a wealth of information on the vibrational and translational product distributions, which can be

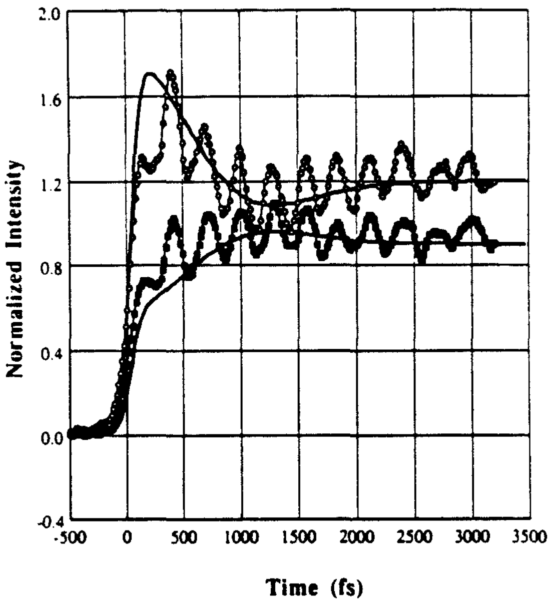


FIG. 18. FTS transients taken from $\lambda_1 = 310$ nm and $\lambda_2 = 390$ nm and detection at $\lambda_{det} = 427.5$ nm. The data is shown after normalization of the asymptotes to 1.2 and 0.9 for parallel (open circles) and perpendicular (filled squares) transients, respectively. The theoretical curves were generated using formulas derived in Ref. 8(b). The important feature to note is the complementary change of the transients when the polarization is changed; the oscillatory behavior is, however, unchanged. (See also Fig. 6.)

made to agree well with experimental observations by an appropriate choice of the PES. The quantum simulation of HgI fluorescence resulting from product molecules with well-defined vibrational distributions is as rich with structure as the observed transients.

FTS of dispersed molecular fluorescence, together with polarization methods, now make it possible to study these elementary reactions with multidimensional PES's. The femtosecond dynamics in such systems reflects the nature of the transition state and the different trajectories leading to products. It is therefore natural to extend these studies to a number of similar reactions in an attempt to map out the surface dynamics.

ACKNOWLEDGMENT

This work was supported by a grant from the Air Force Office of Scientific Research.

[1] A. H. Zewail, Science **242**, 1645 (1988); and referenced therein.
[2] A. H. Zewail and R. B. Bernstein, Chem. Eng. News **66**, 24 (1988).
[3] M. J. Rosker, M. Dantus, and A. H. Zewail, J. Chem. Phys. **89**, 6113 (1988).
[4] M. Dantus, M. J. Rosker, and A. H. Zewail, J. Chem. Phys. **89**, 6128 (1988), and references cited therein.
[5] R. B. Bernstein and A. H. Zewail, J. Chem. Phys. **90**, 829 (1989).
[6] T. S. Rose, M. J. Rosker, and A. H. Zewail, J. Chem. Phys. **91**, 7415 (1989).
[7] T. S. Rose, M. J. Rosker, and A. H. Zewail, J.Chem. Phys. **88**, 6672 (1988); M. J. Rosker, T. S. Rose, and A. H. Zewail, Chem. Phys. Lett. **146**, 175 (1988).
[8] (a) R. M. Bowman, M. Dantus, and A. H. Zewail, Chem. Phys. Lett. **156**, 131 (1989); (b) M. Dantus, R. M. Bowman, J. S. Baskin, and A. H. Zewail, *ibid.* **159**, 406 (1989).
[9] (a) J. Maya, J. Chem. Phys. **67**, 4976 (1977); IEEE J. Quantum Electron. **QE-15** 579 (1979); (b) K. Wieland, Helv. Phys.Acta **2**, 46 (1929); **14**, 420 (1941); K. Wieland, Z. Elektrochem. **64**, 761 (1960); K. S. Viswanathan and J. Tellinghuisen, J. Mol. Spectrosc. **98**, 185 (1983); (c) F. M. Zhang, D. Oba, and D. W. Setser, J. Phys. Chem. **91**, 1099 (1987).
[10] (a) H. Hofmann and S. R. Leone, J. Chem. Phys. **69**, 3819 (1987); (b) J. A. McGarvey, Jr., N. H. Cheung, A. C. Erlandson, and T. A. Cool, *ibid.* **74**, 5133 (1981); N. H. Cheung and T. A. Cool, J. Quant. Spectrosc. Radiat. Transfer **21**, 397 (1979).
[11] For reviews see R. Bersohn, J. Phys. Chem. **88**, 5145 (1984); J. P. Simons, *ibid.* **88**, 1287 (1984); and references therein.
[12] T. M. Mayer, B. E. Wilcomb, and R. B. Bernstein, J. Chem. Phys. **67**, 3507 (1977); T. M. Mayer, J. T. Muckerman, B. E. Wilcomb, and R. B. Bernstein, *ibid.* **67**, 3522 (1977); R. B. Bernstein, *Chemical Dynamics via Molecular Beam and Laser Techniques* (Oxford University, New York, 1982), p. 31; B. E. Wilcomb, T. M. Mayer, R. B. Bernstein, and R. W. Bickes, Jr., J. Am. Chem. Soc. **98**, 4676 (1976).
[13] R. L. Fork, B. I. Greene, and C. V. Shank, Appl. Phys. Lett. **38**, 671 (1981).
[14] R. L. Fork, C. V. Shank, and R. T. Yen, Appl. Phys. Lett. **41**, 223 (1982).
[15] A. H. Zewail, J. Chem. Soc. Faraday Trans. 2, **85**, 1221 (1989).
[16] JANAF Thermochemical Tables, Natl. Stand. Ref. Data Ser. Natl. Bur. Stand. 37, 2nd ed. (1971).
[17] W. R. Wadt, J. Chem. Phys. **72**, 2469 (1980).
[18] R. D. Levine and R. B. Bernstein, *Molecular Reaction Dynamics and Chemical Reactivity* (Oxford University, Oxford, 1987).
[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes* (Cambridge University, Cambridge, 1986).
[20] B. R. Johnson, J. Chem. Phys. **67**, 4086 (1977).
[21] M. Gruebele and A. H. Zewail (in preparation); preliminary results of the full quantum calculations accurately reproduce the FTS transients and the vibrational distributions obtained by classical trajectory calculations.
[22] Similar polarization effects were observed when the probe wavelength was tuned off resonance to 620 nm, where the probing occurred solely in the transition-state region. We observe changes in the relative intensities of the I and I* channels with polarization. We shall use this in future studies to distinguish the different torques in the I and I* channels.

Chapter IV

Applications of FTS

3. Femtosecond transition-state spectroscopy of iodine: From strongly bound to repulsive surface dynamics

Volume 161, number 4,5 CHEMICAL PHYSICS LETTERS 22 September 1989

FEMTOSECOND TRANSITION-STATE SPECTROSCOPY OF IODINE: FROM STRONGLY BOUND TO REPULSIVE SURFACE DYNAMICS

R.M. BOWMAN, M. DANTUS and A.H. ZEWAIL
Arthur Amos Noyes Laboratory of Chemical Physics [1], *California Institute of Technology, Pasadena, CA 91125, USA*

Received 15 August 1989

The application of femtosecond transition-state spectroscopy (FTS) to molecular iodine is reported. The real-time motion of wave packets prepared coherently in the bound B state is observed. In addition, the motion is probed near and above the dissociation limit for the reaction: $I_2 \rightarrow I(^2P_{3/2}) + I^*(^2P_{1/2})$. FTS measurements of the dynamics on repulsive surfaces are also reported.

1. Introduction

In femtosecond transition-state spectroscopy (FTS) [1] of reactions, the evolution of free fragments as they pass through transition states is probed at different interfragment separations (R). The wavelength of the probe pulse (λ_2) can either be in resonance with the free fragment transition (i.e. "infinite" separation R; λ_2^∞) or in tune with the transition state (i.e. fixed R; λ_2^*) [2]. In either case, the total energy available for the reaction ($E_{\rm avl} = \frac{1}{2}\mu v_r^2$, where v_r is the recoil velocity) is determined by the wavelength (λ_1) of the initiation pulse. For a number of reactions this method of probing has resulted in the observation of the motion of the wave function (or wave packet) along unbound surfaces ("purely" repulsive) [1,3], quasi-bound surfaces (avoided crossings) [4], and multidimensional (vibrational) surfaces with a saddle point [5].

The study of halogens is of great interest because they exhibit the above characteristics (repulsive, quasibound and bound surfaces) in the various PES of their reactions:

$$X_2 \rightarrow X + X \quad \text{or} \quad X_2 \rightarrow X + X^* .$$

They also were part of the very early study of crossed-beam reactions [6] and photon/molecular beam reactions [7], and connections with these studies will be valuable.

[1] Contribution No. 8016.

This Letter is concerned with FTS studies of iodine dissociation, and with the wave packet motion in the B state. By tuning λ_1, we observe very strong oscillatory behavior characteristic of the motion in the B state. From this temporal data we can deduce the spectroscopic constants. At other λ_1's we observe rapid decays (on the femtosecond time scale) that are characteristic of the "unbound" states which produce iodine atoms. By varying λ_2^* we probe different parts along R of the potential. The iodine molecule presents a unique opportunity for several reasons: (a) the bound and repulsive potentials have been assigned [8], (b) there is an abundance of spectroscopic information [9], and (c) iodine consists of two identical heavy atoms.

The idea of the experiment on iodine, together with an approximate display of the PESs, are outlined in fig. 1. A pump pulse (λ_1) prepares a coherent wave packet in any of three regions: (a) on a repulsive PES, (b) on a bound PES, or (c) in the continuum (or quasicontinuum) states where dissociation occurs. The probing of the dynamics in the three regions is carried out by the second pulse (λ_2) which takes the coherently prepared wave packet into an upper fluorescent state. The observed signal is a plot of the probe-induced fluorescence as a function of the time delay between the pump and probe lasers. One major advantage of this scheme is the ability to probe at short R's because of the involvement of a bound "fluorescent" state at λ_2^*. This is similar to the earlier case [10] where multiphoton ionization was utilized

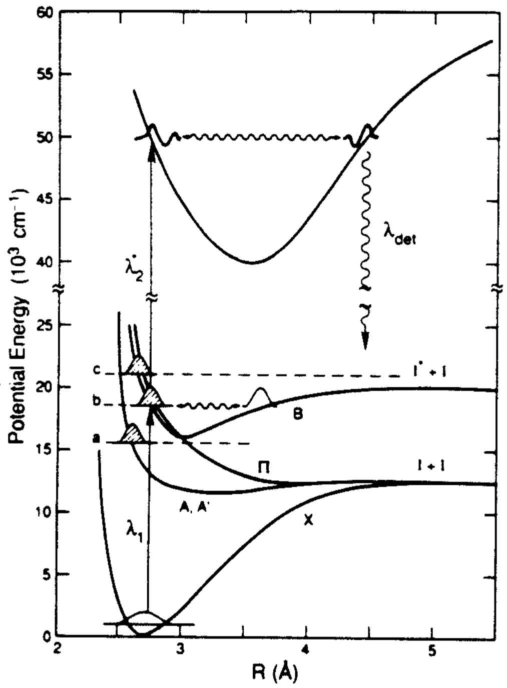


Fig. 1. Schematic (approximate) of the bound and repulsive potential energy surfaces relevant to the FTS study of iodine. The states are labelled briefly, and grouped in accordance with previous spectroscopic work (see text).

to probe dissociation on repulsive surfaces but with the probe tuned to a bound ion state to restrict the range of R.

2. Experimental

The experimental apparatus has been described in detail elsewhere [3–5]. Briefly, the system consists of a CPM laser amplified by a Nd:YAG laser. These amplified pulses have an energy of up to 0.3–0.5 mJ and a pulsewidth (fwhm) as short as 50 fs. Nonlinear techniques (e.g. frequency doubling, sum frequency mixing with the YAG wavelengths and continuum generation) were used to generate the necessary colors for the experiments. All the pulses were spectrally and temporally characterized, and attenuated to insure linearity (or near linearity) of the power dependence. Cross-correlations of 80–90 fs were obtained for the experiments involving the $\approx$ 390 nm mixed light. The iodine was kept in a slow flow fluorescence cell at low pressures ($<$ 100 mTorr) at room temperature.

3. Results and discussion

3.1. Bound state dynamics

Fig. 2 shows a typical transient obtained for a pump wavelength of 620 nm and a probe wavelength of 310 nm detecting the fluorescence at the well known emission (see fig. 1) near 340 nm following single and two photon excitation [8,11]. In the notation of previous publications [5] we designate this transient λ_1/λ_2^* (λ_{det}) in nm=620/310 (340). This transient consists of strongly interfering oscillations. Since the femtosecond pulse duration is shorter than the vibrational period of I_2, the pump prepares the B state of iodine in a coherent superposition of a few vibrational states (determined by the bandwidth of the pulse and the slope of the repulsive surface). Thus, the signal is clearly the result of the interferences between the vibrational levels excited. The short time oscillation (300 fs) is the period of the wave packet motion at this particular energy. It can be directly related to the vibrational levels excited. In this case, $\lambda_1 = 620$ nm and the B state is prepared around $v' = 3$. Using the spectroscopic parameters of I_2 [12], we obtain a vibrational spacing of 122 cm^{-1}, which translates to a period of 273 fs. Better agreement with the experimentally obtained periods is achieved when the initial thermal energy is taken into account; the absorption from the different v'' states is known at the wavelengths studied [12]. Also, the measured period (T_a) is obtained from direct peak to peak measurement and not from complete analysis of the waveform. The shape of the oscillatory transient in the top of fig. 2 can easily be simulated using the conditions of our experiments; the behavior is described simply by the motion of, for example, two wave packets that start in phase and rephase at later times determined by the difference of their frequencies. This analysis together with a Fourier transform treatment of the data will be published elsewhere when these studies are completed.

Keeping the probe wavelength fixed at 310 nm and varying the pump wavelength within the bound por-

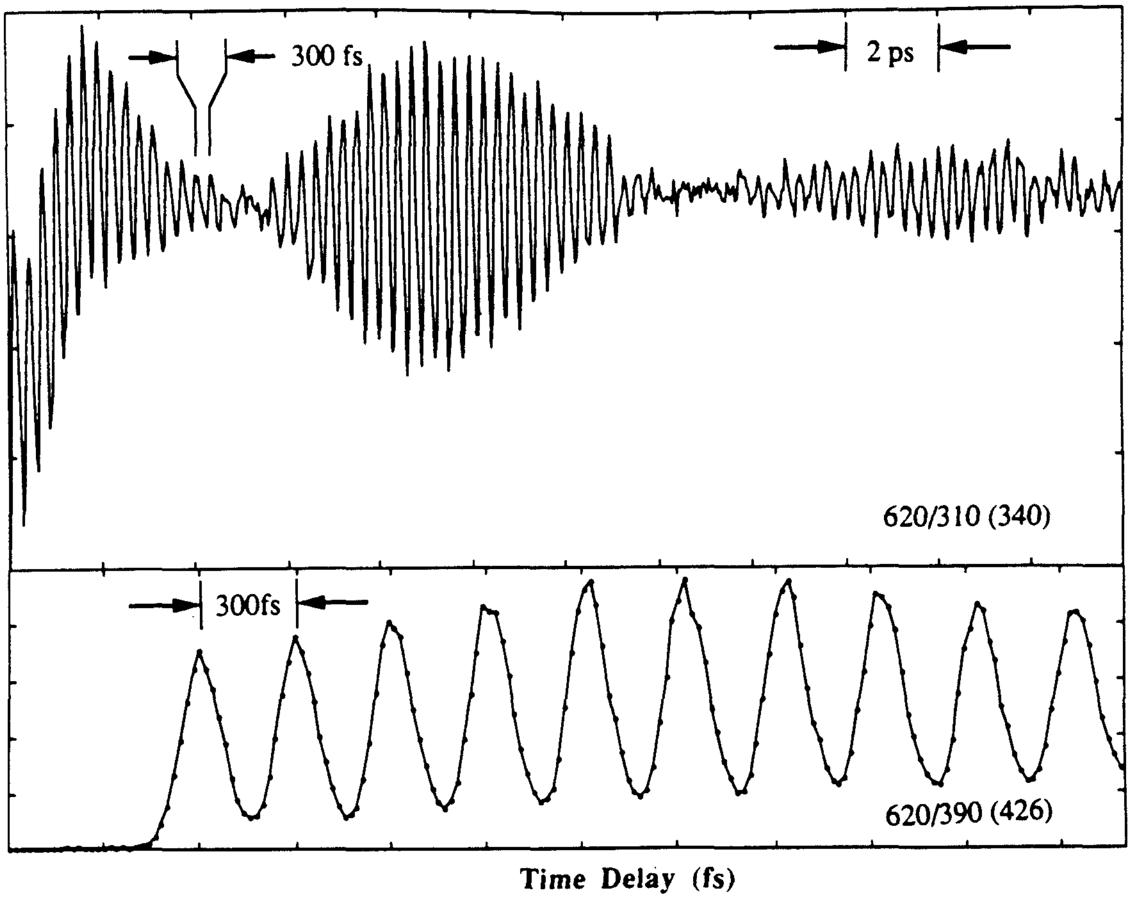


Fig. 2. Top: Typical FTS transient obtained for 620/310 (340) showing the wave packet dynamics in the strongly bound B state. Bottom: Typical FTS transient, 620/390 (426), showing the period of oscillation to be invariant with the probe wavelength. Note that the time scales are different for the two transients.

tion of the B state (634 to 500 nm), i.e. starting at different points in the vibrational manifold, we observe different periods in the coherent evolution of wave packets. Fig. 3 shows a selection of the transients we have recorded where the period of oscillation is shown to increase as the pump energy is increased, i.e. motion starting at higher vibrational levels.

The transients in fig. 3 show oscillations with an average period of 750, 500 and 290 fs for the top (this transient shows an irregular oscillation pattern at times longer than shown), middle and bottom transients, respectively. Calculation of these oscillation periods has also been made using the available spectroscopic information of the B state, and good agreement was found between the calculated and observed periods as a function of excitation energy. In the future we will use the data to deduce characteristics of the potential surface in the region of dissociation to I+I*.

In order to confirm that the oscillations are due to the wave packet prepared by the pump pulse, we used a different probe pulse at 390 nm. In this case the fluorescence maxima occurred at 426 nm, known as

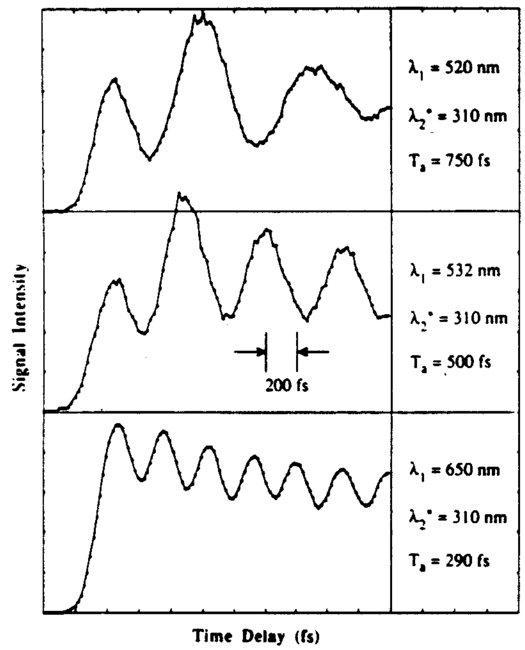


Fig. 3. FTS transients obtained at different pump laser wavelengths (λ_1). The time scale for the three traces is the same (200 fs/division). The periods are measured peak to peak between the second and third oscillations in order to avoid convolution effects. Complete analysis of these transients will give better values (see text).

the E→B transition [13]. The bottom of fig. 1 shows a typical 620/390 (426) transient where the short time oscillation period is the same as that of the 620/310 (340) transient, i.e. 300 fs. It should be noted that the depth of modulation and envelope decay are different for the two probe wavelengths. This can be related to the position in the well where the Franck–Condon overlap with respect to the probe pulse is maximized, and this will be analyzed in our forthcoming study.

The measured oscillations of the wave packet prepared in this fashion last for longer than 40 ps. There is no sign of decay of the total signal since the lifetime of the B state is very long (microsecond time scale) compared to the femtosecond time scale of these experiments. However, the envelope of the transients decays on a shorter time scale (see fig. 2) and this reflects the coherence decay dictated by the preparation process and the spread of the packets.

3.2. Repulsive versus bound surfaces

In the previous section the pump pulse accessed states within the bound portion of the B state PES and showed well behaved oscillations. We now turn to experiments where the pumping is into the continuum levels above the dissociation limit to I+I* (higher energies), and pumping into known repulsive states at lower energies than the B state. These data are shown in fig. 4.

The top transient in fig. 4 was obtained by pumping at 700 nm. The absorption spectrum of I_2 can be resolved into three regions for the three states involved: $^1\Pi_{1u}$ (blue absorption), B $^3\Pi_{0u}$ and A $^3\Pi_{1u}$ (red absorption) [12]. The bottom of the B state well corresponds to 634 nm, and the absorption at

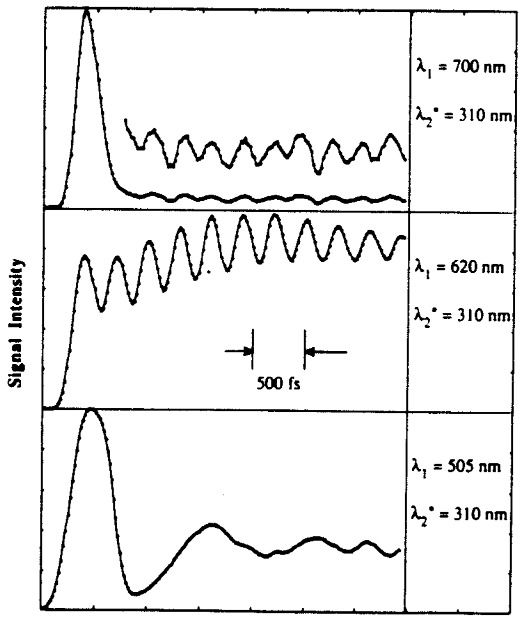


Fig. 4. FTS transients obtained using different pump energies spanning repulsive and bound states. Top: Transient corresponding to case (a) type in fig. 1. At long times the oscillatory behavior is displayed. Middle: Transient obtained from strongly bound B state (see fig. 1). Bottom: Transient showing the time evolution of a wave packet near the B state dissociation limit.

700 nm, therefore, is due to the A state and some hot band absorption to the B state [12]. The 700/310 (340) transient shows a rise and decay typical of FTS off-resonant probing. When the latter part of the transient is expanded (×5), oscillations with a period of 250 fs are seen (the amplitude of the oscillating portion of the signal corresponds to ≈5% of the total signal). Near the bottom of the B state this period is expected. The observed FTS transient, therefore, reflects the dynamics in this region for both the A and B states.

Tuning the pump laser to lower energies minimizes the overlap with the B state. The transient, 750/310 (340) (not shown) displays totally FTS off-resonance behavior as has been discussed in great detail in previous works [1-5]. The signal shows a rise and decay much like the 700/310 (340) data at the top of fig. 4, but the signal decays to zero amplitude showing no oscillatory behavior. The information contained in this transient corresponds to the evolution of the wave packet at very short R's in the Franck-Condon region, and we are in the process of analyzing the time-dependent shape.

The upper limit of the B state presents a different opportunity to explore dissociative states but this time the continuum or quasicontinuum states (case (c) in fig. 1). In addition the $^1\Pi$ has an absorption at these higher energies. Tuning the pump wavelength to 505 nm we obtained the transient shown in the bottom of fig. 4. Unlike the transients in fig. 2 or the transient in the middle of fig. 4, this transient shows a very large signal at early times and damped oscillations. From the latter portion of the 505/310 (340) data it is concluded that some fraction of the wave packet is reflected and returns to the probing region after ≈1.2 ps. The highly anharmonic vibrational levels of the B state ($v' \approx 50$) gives a period similar in value.

The reflected portion of the wave packet corresponds to the quasicontinuum states overlapped by the pump laser. Tuning to higher energies (490 nm) insures no bound states are overlapped and no wave packet reflection is observed. The 490/310 (340) transient (not shown) simply displays FTS off-resonance-like behavior, which in this case is a combination of both the continuum B state and the purely repulsive $^1\Pi$ state dynamics.

From these transients at different λ_1 and λ_2^*, characteristics of these PESs can be deduced in the bound and dissociative regions. The transients shown in figs. 3 and 4 are ideal for the inversion method given in ref. [14]. In fact, the data in fig. 4 (bottom) are reminiscent of the expected behavior [14] for FTS observations on a repulsive surface with a well. This analysis will be given elsewhere.

Finally, as for the FTS experiments of HgI_2 [5], we have observed polarization anisotropy decays on the femtosecond time scale for I_2, and this should give the alignment information, similar to that obtained in ref. [5]. In this case, analysis of the alignment in real-time will give a comparison of the rotational (angular) dephasing time to the scalar wave packet dynamics, i.e. the period of oscillation or the dissociation time. In all experiments reported here the relative polarization was fixed at perpendicular.

4. Conclusions

The application of FTS to molecular iodine has allowed us to observe the real-time motion of wave packets prepared coherently in a bound, quasicontinuum or repulsive state. For the B state, FTS shows that the wave packet moves in the well, and dephases and rephases with well-defined time constants, directly related to the vibrational energies and anharmonicity. Since the spectroscopy of I_2 is well known, comparisons with the B state potential are made. By tuning the pump wavelength to the red of the potential minimum and to the blue of the dissociation limit we were able to probe the dynamics on both a dissociative potential and in the continuum levels of a bound PES. In all of these cases the probing was done in the region of small interfragment separation (R) accessed in the Franck-Condon region of the pump pulse. Probing at larger R will involve the detection of iodine atoms as done in other FTS studies [1-5]. In progress are experiments on other halogens and interhalogens [15]. We are also performing condensed phase experiments to understand the influence of solvation on these bound and unbound systems using FTS.

Acknowledgement

This work was supported by a grant from the Air Force Office of Scientific Research.

References

[1] M. Dantus, M.J. Rosker and A.H. Zewail, J. Chem. Phys. 87 (1987) 2395.
[2] A.H. Zewail, Science 242 (1988) 1645;
A.H. Zewail and R.B. Bernstein, Chem. Eng. News 66 (1988) 24;
J. Baggott, New Scientist 1669 (June 17, 1989) 58.
[3] M.J. Rosker, M. Dantus and A.H. Zewail, J. Chem. Phys. 89 (1988) 6113;
M. Dantus, M.J. Rosker and A.H. Zewail, J. Chem. Phys. 89 (1988) 6128.
[4] T.S. Rose, M.J. Rosker and A.H. Zewail, J. Chem. Phys. 88 (1988) 6672;
M.J. Rosker, T.S. Rose and A.H. Zewail, Chem. Phys. Letters 146 (1988) 175;
T.S. Rose, M.J. Rosker and A.H. Zewail, J. Chem. Phys., in press.
[5] R.M. Bowman, M. Dantus and A.H. Zewail, Chem. Phys. Letters 156 (1989) 131;
M. Dantus, R.M. Bowman, J.S. Baskin and A.H. Zewail, Chem. Phys. Letters 159 (1989) 406;
M. Dantus, R.M. Bowman, M. Gruebele and A.H. Zewail, J. Chem. Phys., in press.
[6] D.R. Herschbach, Faraday Discussions Chem. Soc. 55 (1973) 233;
R.N. Zare and D.R. Herschbach, Proc. IEEE 51 (1963) 173; Appl. Opt. Suppl. 2 (Chemical Lasers) (1965) 193.
[7] G.E. Busch, R.T. Mahoney, R.I. Morris and K.R. Wilson, J. Chem. Phys. 51 (1969) 837;
R.J. Oldman, R.K. Sander and K.R. Wilson, J. Chem. Phys. 54 (1971) 4127;
R.W. Diesen, J.C. Wahr and S.E. Adler, J. Chem. Phys. 50 (1969) 3635.
[8] R.S. Mulliken, J. Chem. Phys. 55 (1971) 288;
J.C.D. Brand and A.R. Hoy, Appl. Spectry. Rev. 23 (1987) 285.
[9] J.I. Steinfeld, R.N. Zare, L. Jones, M. Lesk and W. Klemperer, J. Chem. Phys. 42 (1965) 25;
R.J. LeRoy and R.B. Bernstein, J. Mol. Spectry. 37 (1971) 109;
M.D. Danyluk and G.W. King, Chem. Phys. 25 (1977) 343;
A.L. Guy, K.S. Viswanathan, A. Sur and J. Tellinghuisen, Chem. Phys. Letters 73 (1980) 582.
[10] L.R. Khundkar and A.H. Zewail, Chem. Phys. Letters 142 (1987) 426.
[11] H. Hemmati and G.J. Collins, Chem. Phys. Letters 75 (1980) 488;
U. Heeman, H. Knöckel and E. Tiemann, Chem. Phys. Letters 90 (1982) 17;
J.C.D. Brand and A.R. Hoy, Can. J. Phys. 60 (1982) 1209;
H.P. Grieneisen and R.E. Francke, Chem. Phys. Letters 88 (1982) 585.
[12] J. Tellinghuisen, J. Chem. Phys. 58 (1973) 2821;
P. Luc, J. Mol. Spectry. 80 (1980) 41;
S. Gerstenkorn and P. Luc, Atlas du spectre d'absorption de la molécule d'Iode (CNRS, Paris, 1978);
R.B. Snadden, J. Chem. Educ. 64 (1987) 919.
[13] D.L. Rousseau and P.F. Williams, Phys. Rev. Letters 33 (1974) 1368;
D.L. Rousseau, J. Mol. Spectry. 58 (1975) 481;
M.D. Danyluk and G.W. King, Chem. Phys. 22 (1977) 59;
S.L. Cunha, J. A. Lisboa, R.E. Francke and H.P. Grieneisen, Opt. Commun. 28 (1979) 321;
G.W. King, I.M. Littlewood and J.R. Robins, Chem. Phys. 56 (1981) 145;
J.C.D. Brand, A.K. Kalukar and A.B. Yamashita, Opt. Commun. 39 (1981) 235.
[14] R.B. Bernstein and A.H. Zewail, J. Chem. Phys. 90 (1989) 829.
[15] M.S. Child and R.B. Bernstein, J. Chem. Phys. 59 (1973) 5916;
J.C.D. Brand and A.R. Hoy, Appl. Spectry. Rev. 23 (1987) 285.

Chapter IV

Applications of FTS

4. Femtosecond real-time alignment in chemical reactions

Volume 159, number 5,6 CHEMICAL PHYSICS LETTERS 21 July 1989

FEMTOSECOND REAL-TIME ALIGNMENT IN CHEMICAL REACTIONS

M. DANTUS, R.M. BOWMAN, J.S. BASKIN and A.H. ZEWAIL

Arthur Amos Noyes Laboratory of Chemical Physics [1], California Institute of Technology, Pasadena, CA 91125, USA

Received 25 April 1989; in final form 5 June 1989

Femtosecond real-time alignment in dissociation reactions is observed experimentally for HgI_2 and ICN. The results are accounted for by the classical theory outlined here. This study adds a new dimension to femtosecond transition-state spectroscopy (FTS), namely the probing of the "vector dynamics" of reactions.

1. Introduction

In probing the femtosecond (and picosecond) dynamics of reactions [1-3] the major focus thus far has been on studies of the (scalar) recoil of fragments along the reaction coordinate. More recently, an elementary theoretical description [4] has been made that addresses the time evolution of (vector) alignment and the angular momentum of the fragments. Using femtosecond transition-state spectroscopy (FTS) [1-3] but with polarized pulses, the alignment during dissociation can be measured and then related directly to the final rotational distribution of the fragments [4]. This effect of rotation, which is similar to purely rotational coherence in photoexcitation experiments [5], can be manifested directly in the FTS, and as shown by Metiu's group [6] and Wilson's group [7] for ICN dissociation, can be rigorously calculated using model potentials.

In this Letter, we present the applications of a simplified theory [4] of time-dependent alignment in FTS experiments to two classes of reactions. The simplest systems for observation of alignment during dissociation are of the type

$$ABC^* \rightarrow [ABC]^{\ddagger *} \rightarrow AB + C\,.$$

If the AB fragment experiences a torque during dissociation, it begins to rotate and acquires angular momentum. It is of great interest to attempt probing this time-dependent angular momentum [#1] in real time, and to relate the observables to the nature of the force field along the reaction coordinates: internuclear separation and angle of rotation of the AB fragment.

To explore these ideas we studied the femtosecond alignment in the following two reactions:

$$I\,CN^* \rightarrow [I\,CN]^{\ddagger *} \rightarrow CN + I$$

and

$$I\,Hg\,I^* \rightarrow [I\,Hg\,I]^{\ddagger *} \rightarrow HgI + I\,.$$

For both reactions the dynamics involving the direct separation of the fragments in real time [9,10], and the time-integrated alignment [11-14] have been studied. As discussed in refs. [9,10] there have been a number of studies of product-state distributions that are relevant to our FTS work. Here, we focus our attention on observations related to the time-dependent alignment in the course of dissociation. These observations are analysed according to a theoretical description [4] which considers the evolution of the dissociation as a coherent superposition of aligned fragments. A direct comparison is made between the rotation of the CN fragment and the HgI fragment. Due to the large difference in mass, the alignment occurs on different time scales for these fragments. The experiments allow us to add a new dimension to FTS, namely the "vector dynamics" of the reaction.

[1] Contribution No. 7936.

[#1] For an excellent discussion of angular momentum and alignment see ref. [8].

0 009-2614/89/$ 03.50 © Elsevier Science Publishers B.V. (North-Holland Physics Publishing Division)

We hope to extend these studies to other reactions [15,16] for reasons that will become clear below.

2. Experimental

The experimental apparatus has been described in detail elsewhere [1,9,10]. Briefly, the system consists of a CPM laser amplified at 20 Hz by a Nd:YAG laser. These amplified pulses have an energy of 0.3–0.5 mJ and a pulsewidth (fwhm) as short as 50 fs. Standard nonlinear techniques (e.g. frequency doubling and sum frequency mixing with the YAG wavelengths) were used to generate the necessary colors to do the alignment experiments. All the pulses were spectrally and temporally characterized, and attenuated to insure linearity (or near linearity) of the power dependence. Cross-correlations of 80–90 fs were obtained for the experiments involving the ≈ 390 nm mixed light. The polarization of all of the pulses was checked to insure high extinctions.

Except for the polarization aspects, the details of these FTS experiments are identical to the ones described in previous femtochemistry works and will not be repeated in detail here. After a polarized pump initiated the dissociation, the fragments were then detected with a polarized probe pulse of the appropriate color. The probe pulse was polarized either perpendicular or parallel to the pump pulse polarization. The DEA/MPI technique [1,9] was used to establish the $t=0$ position for both polarization cases. The conditions for the ICN and HgI_2 experiments were the same as those for the previous femtochemistry studies [9,10].

3. Theory

Here, we shall consider a classical treatment of product alignment in prompt, impulsive dissociation of triatomics [4]. The reaction is initiated by a linearly polarized laser pulse, and an electronic transition of the molecular fragment is probed by a time-delayed pulse having linear polarization either parallel or perpendicular to the pump. The reagent population created by the pump pulse from the initially isotropic sample has a well-defined spatial orientation determined by a $\cos^2\theta$ distribution of transition dipoles of the reagent molecules with respect to the pump polarization vector. The probability for probe absorption will vary with time and with probe polarization as the nascent molecular fragments undergo rotation, carrying the molecule-fixed probe transition dipoles with them. Fluorescence from the state reached via the probe transition is detected, providing a signal which is assumed proportional to the population in that state [#2] (vide infra).

By assuming that the separation of product fragments is complete on a time scale much shorter than the rotational motion, the evolution of the dipole alignment and hence of each fluorescence signal ($I_\parallel$ and $I_\perp$) can be described completely in terms of free rotation of the nascent fragments. This and other cases will be detailed elsewhere [17]. The evolution of the dipole orientation is, thus, identical to that following excitation of an isotropic sample of stable isolated molecules. When such a sample is excited by a linearly polarized light pulse, it can be shown that the signal polarization anisotropy

$$r(t)=(I_\parallel-I_\perp)/(I_\parallel+2I_\perp)\,, \tag{1a}$$

for all molecules characterized by angular momentum j, is given by

$$r(t)=0.4\langle P_2(\cos\eta(t))\rangle\,, \tag{1b}$$

where $\eta(t)$ is the angle between the transition dipole of the pump transition at $t=0$ and that of the probe at time t (details will be given in ref. [17]). The ensemble average must account for all possible orientations of the vector j in the molecule fixed frame at $t=0$. To specialize this result to the situation currently under investigation, we let $\eta(t)$ be the angle between reagent dipole at $t=0$ and product dipole at time t. Both dipoles are assumed for the cases of interest to coincide with the internuclear axis ($\hat{z}$) of

[#2] While the total fluorescence intensity is quite generally proportional to the emitting population, the intensity measured in a given direction and/or with a given polarization may show anomalies related to the macroscopic orientation of the emitting dipoles (as, e.g., the well known decay of $I_\parallel$ and buildup of $I_\perp$ for a polarized single pulse experiment). In a gas phase sample, a substantial degree of orientation anisotropy may persist indefinitely, so that even time-integrated fluorescence is not necessarily proportional to the excited population for arbitrary detection geometry.

the diatomic fragment, whose motion is that of the figure axis of a symmetric top:

$$\hat{z}(t) = \sin\theta \sin(\omega t)\,\hat{X} - \sin\theta\cos(\omega t)\,\hat{Y} + \cos\theta\,\hat{Z}, \qquad (2)$$

described in terms of a lab fixed frame with $\hat{Z}$ chosen along j. $\omega = 4\pi Bj$ is the nutation frequency of the top with rotational constant B, angular momentum j, and θ is the angle between the top axis and j. Consequently:

$$\cos\eta(t) = \hat{z}(0)\cdot\hat{z}(t) = \cos^2\theta + \sin^2\theta\cos(\omega t), \qquad (3)$$

where the only dependence on the orientation of j is through the angle θ. For a diatomic, N, the angular momentum of nuclear rotation, is perpendicular to the internuclear axis. For most molecules of the sample $j \approx N$, so θ is very close to $\pi/2$, and to a good approximation, $\cos\eta(t) = \cos(\omega t)$. The average indicated in eq. (1b) is unimportant in this approximation, and

$$r(j,t) = 0.4\,P_2(\cos\eta(t))$$
$$= \tfrac{3}{5}\cos^2(\omega t) - \tfrac{1}{5} = 0.1 + 0.3\cos(2\omega t). \qquad (4)$$

Thus, the anisotropy of the signal from molecules in a particular rotational level oscillates at a frequency fixed by its angular momentum [4].

The macroscopic anisotropy is obtained from those of individual product rotational levels by a summation over the product state distribution $P(j)$:

$$r(t) = \frac{\sum_j P(j) r(j,t)}{\sum_j P(j)}. \qquad (5)$$

This distribution is either obtained experimentally or in some cases prescribed, e.g., as Gaussian in nature [4].

Direct comparison to experiment can be made only after the effects of finite temporal response and population evolution ($A(t)$) have been accounted for in the individual intensity measurements. The intensities are simply recovered from the theoretical anisotropy using the definition of $r(t)$ and the fact that $I_\parallel + 2I_\perp \propto A(t)$. Thus:

$$I(t) \propto [1 + \alpha r(t)]\,A(t), \qquad (6)$$

where α takes the value of 2 for parallel detection, and -1 for perpendicular detection. The functional forms used for $A(t)$ depend on the probe wavelength and bandwidth and have been described elsewhere [4].

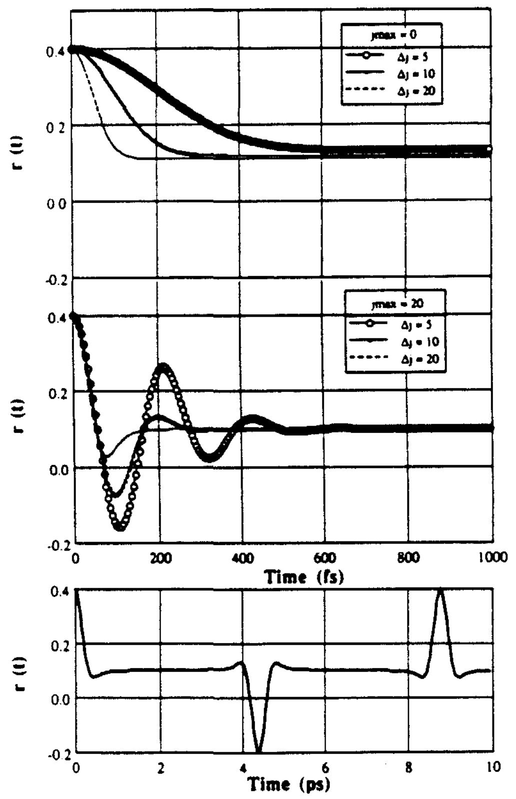


Fig. 1. Theoretical simulations of $r(t)$ using the formulas presented in the text. These simulations illustrate the effect of the rotational distribution of the coherent ensemble. The rotational distribution is taken to be a Gaussian centered at j_{max} with a half-width Δj (at $1/e$). The top figure shows the effect of broadening the rotational distribution (Δj). The middle figure shows the same effect but for a distribution centered at $j = 20$. Oscillations are observed at a period that corresponds to $j = 20$. The bottom figure corresponds to a population characterized by a Gaussian centered at $j = 2$ and with $\Delta j = 5$. Note in this case that the time scale is 10 times longer and that recurrences are observed.

In fig. 1, $r(t)$ is computed for different fragment rotational distributions. Clearly, the nature and degree of the torque has a direct signature on $r(t)$. If j_{max} is shifted from $j = 0$, oscillations are observed, and the coherence time [4] is directly related to the width of the rotational distribution. For the early time behavior considered here, the choice of classical angular momentum to ascribe to the population characterized by quantum number j is not critical. Behavior at longer times can only be accurately predicted by use of the proper quantum mechanical rotational beat frequencies, however. Thus, for example, when $j\hbar$ is taken as the classical angular momentum of the j state, a full in-phase recurrence is predicted by the above treatment at $t=(2\pi/2\omega)|_{j=1}=1/4B$, while an out-of-phase recurrence is actually expected quantum mechanically. In- and out-of-phase recurrences have been observed in LIF [5] and in pump–probe experiments [18] in our laboratories, and recently in fluorescence depletion [19] by McDonald's group. These types of recurrences should also be observed for photodissociation fragments when the delay time reaches $1/4B$. In this simple picture, an out-of-phase recurrence for CN and HgI should be at 4.39 and 304 ps and hence an in-phase recurrence at 8.77 and 608 ps, respectively. The experimental data shown here are made at early times and are of particular interest since they carry information about the initial dynamics of the dissociation process.

4. Results and discussion

The results in figs. 2 and 3 show $I_\parallel(t)$ and $I_\perp(t)$ transients for HgI_2 and ICN, where the subscripts correspond to the relative polarization of the pump and probe pulses. The oscillations in the HgI_2 transients are due to recoil dynamics on the PES and are not directly related to the alignment. These oscillations have been presented in a previous study [10] and will be analyzed more fully in a future publication for all wavelengths of detection recently studied. The asymptote in each case has been normalized to 1.2 or 0.9 for $I_\parallel(t)$ or $I_\perp(t)$ transients, respectively. These values come from the equations presented in the theory section, which can be explicitly written as

$$I_\parallel(t)=[1.2+0.6\cos(8\pi Bjt)]A(t)\,, \qquad (7a)$$

$$I_\perp(t)=[0.9-0.3\cos(8\pi Bjt)]A(t)\,. \qquad (7b)$$

These expressions are identical to those of eq. (6) in ref. [4], and indicate that for a single j oscillatory transients are expected, while for a distribution of j's "dephasing" will occur with a characteristic coherence time, defined as τ_c and determined by Δj (the half-width of the distribution for, e.g., a Gaussian at $1/e$) and by B.

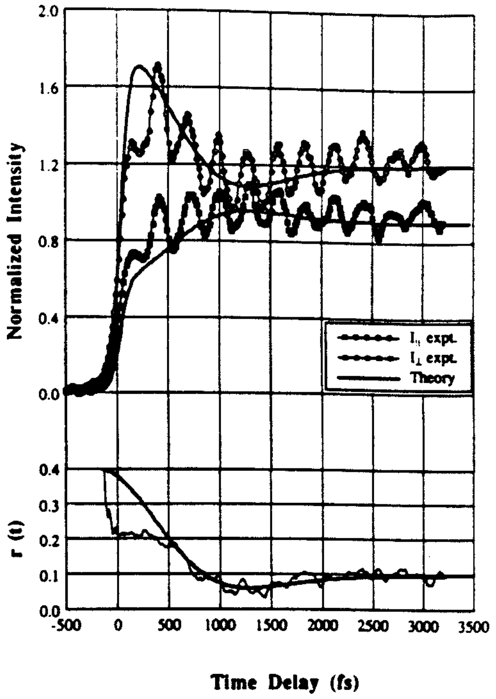


Fig. 2. Top: HgI_2 FTS transients obtained for $\lambda_1=308$ nm, $\lambda_2=390$ nm and detection centered at 427.5 nm. The data are shown after normalization of the asymptotes to 1.2 and 0.9 for parallel and perpendicular transients, respectively. The theoretical curves were generated as described in the text. Bottom: $r(t)$ obtained from the top transients and calculated $r(t)$ (heavy line). Note the manifestations of the time-dependent alignment in both $I_\parallel$ and $I_\perp$ FTS transients, and the good agreement with theory.

As would be expected, the parallel case reaches a higher value and grows in more quickly than the perpendicular case. This is apparent in the HgI_2 transients (fig. 2), taken while the probe is nearly on-resonance with the fragment. Also included in fig. 2 are the transients expected theoretically using eqs. (7a) and (7b). These curves were obtained using a Gaussian rotational distribution with $j_{max}=80$ and $\Delta j=90$. Unlike previous FTS experiments [9], the exact functional form for $A(t)$ in the HgI_2 case is not given because of the two-dimensional recoil dynamics. The $A(t)$ dynamics are treated explicitly by Dantus et al. in ref. [10] using trajectory calculations. Here, as a first approximation for $A(t)$, an on-resonance tran-

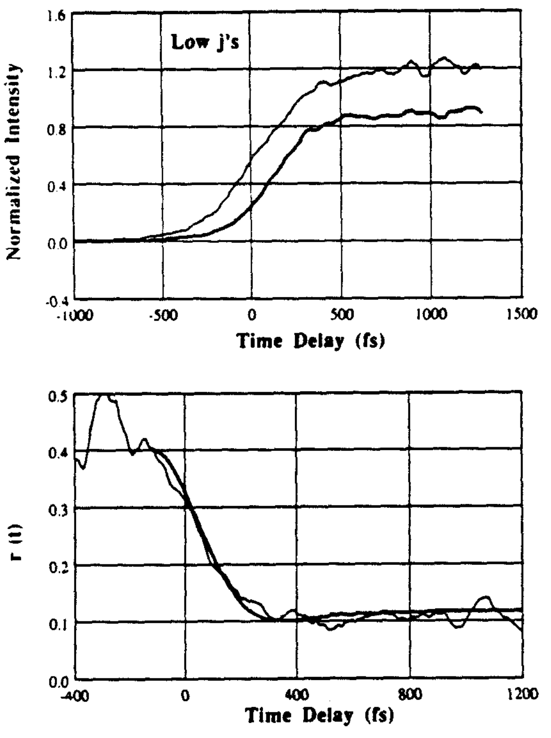


Fig. 3. Top: Parallel (thin line) and perpendicular (heavy line) ICN FTS transients obtained for $\lambda_1 = 305.3$ nm, $\lambda_2 = 387.9$ nm. The data are shown after normalization of the asymptotes to 1.2 and 0.9 for parallel and perpendicular transients, respectively. Bottom: $r(t)$ obtained from the top transients. Calculated $r(t)$ using a Gaussian for the rotational distribution centered at $j=2$ with $\Delta j = 5$ (as estimated by comparing the spectrum of the probe laser and the absorption spectrum of CN).

sient with a highly repulsive excited potential surface was used [9] (both transients were convoluted with a 150 fs response). Since the form of $A(t)$ is only important while the molecules are dissociating, the largest discrepancies will occur at early times. This is seen in fig. 2; the fits become quite good at longer times. Of course this will introduce larger error at early times in the theoretical $r(t)$'s, yet qualitative agreement is found (vide infra).

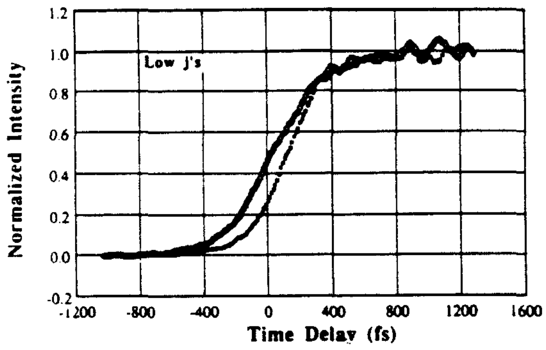
The plots of $r(t)$ along with simulated $r(t)$'s for both HgI_2 and ICN are shown [3] in figs. 2 and 3. For HgI_2 we observe a decay (much longer than the pulse width) and a "dip" that is also evident in the $I_\parallel(t)$ transient! The observation of the dip indicates that the rotational distribution is shifted from $j=0$. Using $j_{max} = 80$ and $\Delta j = 90$ we obtained the fit for $r(t)$ in fig. 2. The coherence time, τ_c, is long ≈ 1 ps. For the CN in the case of ICN, we have $j_{max} = 2$ and $\Delta j = 5$ (see below). In this case τ_c is less than 280 fs.

The anisotropy lasts a much longer time for the HgI fragment than the CN fragment, implying a much longer dephasing (or coherence) time for HgI. This is what one would expect when comparing the fundamental frequencies of the two fragments, $\omega_0 = 0.72$ ps^{-1} for CN and $\omega_0 = 0.01$ ps^{-1} for HgI. For the same value of j, the CN fragments rotate much more quickly (a factor of 72) than the HgI fragment, resulting in a faster dephasing time for a similar j distribution. One point that should be mentioned is that these times (and forms of the $r(t)$ as mentioned below) are very dependent on the j distribution. A highly rotationally excited j distribution for HgI would result in a faster dephasing time. In the future we hope to quantify the nature of the time-dependent growth of the rotational distribution, and hence the angular distribution.

The alignment in the dissociation of ICN can be seen clearly in the results presented in fig. 4. Since the ICN dissociation is on the same time scale of the alignment (≈ 200 fs), low j's and high j's probing show different FTS. For high j's [4], where τ_c is short compared with the recoil time, both $I_\parallel$ and $I_\perp$ are very similar and both are shifted from $t=0$ by the bond breaking time (≈ 200 fs) reported previously [9]. On the other hand, for low j's the recoil time and the rotational time are comparable and a difference is observed which has its maximum very early in time. These results are consistent with the derived $r(t)$ and τ_c for CN. The exact τ_c cannot be quantified

[3] Unpolarized fluorescence was detected in a direction perpendicular to the probe polarization (the pump and probe beams were collinear and propagate perpendicular to the detection axis). Under these conditions a slight difference between the detected signal and the emitting population is expected. We have calculated this effect for the case at hand and found it to be negligible at the level of our signal-to-noise. More details will be presented in ref. [17].

[4] Note that for high j's the levels probed here are in one branch, while in order to monitor the coherence of the dissociation impulse a much broader fs pulse than the one used is needed. On the other hand, for low j's, coherence can be monitored because both branches are accessible to the probe.



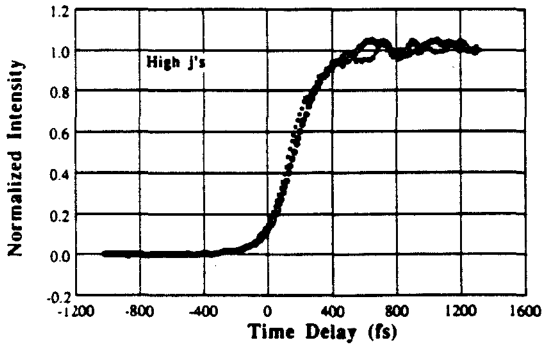


Fig. 4. Top: Parallel (empty circles) and perpendicular (filled squares) ICN FTS transients obtained form probing at $\lambda_2=387.9$ nm (low j's). The transients are normalized to unity in order to show the effect of alignment at early times. Bottom: Parallel (empty circles) and perpendicular (filled squares) ICN FTS transients obtained from probing at $\lambda_2=388.6$ nm (high j's). The transients are normalized to unity. The zero of time has been determined individually for each transient with an accuracy of better than 20 fs. The differences in the rise times for the top and bottom figures can be explained in terms of rotational alignment (see text).

at the moment because we must deconvolute the data near $t=0$ with our cross-correlation response (time resolution 80–90 fs). Thus, our Δj value for ICN is a lower limit. Finally, noise and the precision with which we can determine the $t=0$ point (± 30 fs) for the transients results in experimental $r(t)$'s that carry large uncertainties ($\approx 20\%$). However, it should be noted that the time-dependent alignment is evident in both the $I_\parallel$ and $I_\perp$ FTS transients (see fig. 2). The cases chosen here are representative of the different $r(t)$'s that describe the alignment. More complex $r(t)$'s are quite possible and may reflect the nature of the coupling of the rotational and recoil motions.

With FTS and polarized pulses it is now possible to probe the effects of different rotational distributions, the nature of the torque involved (angular part of the PES) and the recoil dynamics in $A(t)$. We hope to detail the scalar and vector real-time dynamics of these and other elementary reactions in future publications and to relate them to the nature of the PES [20].

Acknowledgement

This work was supported by a grant from the Air Force Office of Scientific Research. We wish to thank Professor Kent Wilson, Professor Horia Metiu and Professor Reinhard Schinke for sending their manuscripts, and for stimulating discussions.

References

[1] A.H. Zewail, Science 242 (1988) 1645, and references therein.
[2] A.H. Zewail and R.B. Bernstein, Chem. Eng. News 66 (1988) 24.
[3] J.L. Knee and A.H. Zewail, Spectroscopy 3 (1988) 44, and references therein.
[4] A.H. Zewail, J. Chem. Soc. Faraday Trans. II 85 (1989), in press.
[5] P.M. Felker, J.S. Baskin and A.H. Zewail, J. Phys. Chem. 90 (1986) 724;
P.M. Felker and A.H. Zewail, J. Chem. Phys. 86 (1987) 2460;
J.S. Baskin, P.M. Felker and A.H. Zewail, J. Chem. Phys. 86 (1987) 2483.
[6] R. Heather and H. Metiu, Chem. Phys. Letters, submitted for publication.
[7] I. Benjamin and K.R. Wilson, J. Chem. Phys., submitted for publication.
[8] R.N. Zare, Angular momentum (Wiley, New York, 1988).
[9] M. Dantus, M.J. Rosker and A.H. Zewail, J. Chem. Phys. 87 (1987) 2395; 89 (1988) 6128;
M.J. Rosker, M. Dantus and A.H. Zewail, J. Chem. Phys. 89 (1988) 6113; Sciences 241 (1988) 1200.
[10] R.M. Bowman, M. Dantus and A.H. Zewail, Chem. Phys. Letters 156 (1989) 131;
M. Dantus, R.M. Bowman, M. Gruebele and A.H. Zewail, J. Chem. Phys., submitted for publication.

[11] M.A. O'Halloran, H. Joswig and R.N. Zare, J. Chem. Phys. 87 (1987) 303;
E. Hasselbrink, J.R. Waldek and R.N. Zare, Chem. Phys. 126 (1988) 191.

[12] W.J. Marinelli, N. Sivakumar and P.L. Houston, J. Phys. Chem. 88 (1984) 6685;
G.E. Hall, N. Sivakumar and P.L. Houston, J. Chem. Phys. 84 (1986) 2120.

[13] I. Nadler, H. Reisler and C. Wittig, Chem. Phys. Letters 103 (1984) 451;
I. Nadler, D. Mahgerefteh, H. Reisler and C. Wittig, J. Chem. Phys. 82 (1985) 3885.

[14] J.H. Ling and K.R. Wilson, J. Chem. Phys. 63 (1975) 101.

[15] T.S. Rose, M.J. Rosker and A.H. Zewail, J. Chem. Phys. 88 (1988) 6672;
M.J. Rosker, T.S. Rose and A.H. Zewail, Chem. Phys. Letters 146 (1988) 175.

[16] N.F. Scherer, L.R. Khundkar, R.B. Bernstein and A.H. Zewail, J. Chem. Phys. 87 (1987) 1451;
N.F. Scherer, C.N. Sipes, R.B. Bernstein and A.H. Zewail, to be published.

[17] J.S. Baskin and A.H. Zewail, to be published.

[18] N.F. Scherer, L.R. Khundkar, T.S. Rose and A.H. Zewail, J. Phys. Chem. 91 (1987) 6478.

[19] M.J. Côté, J.F. Kauffman, P.G. Smith and J.D. McDonald, J. Chem. Phys. 90 (1989) 2865, 2874.

[20] K. Kühl and R. Schinke, Chem. Phys. Letters, submitted for publication.

Appendix 1

Autocorrelation functions of common pulse shapes

Autocorrelation Functions of Common Pulse Shapes

Pulse shape	$I(t)$	$G^2(\tau)$	$\Delta\tau/\Delta t$	$\Delta\nu\,\Delta t$
Square	$1 (-\Delta t/2 \leq t \leq \Delta t/2)$	$1 - \left\vert\frac{\tau}{\Delta t}\right\vert (-\Delta t \leq \tau \leq \Delta t)$	1	0.886
Gaussian	$\exp\left\{-\frac{4\ln 2}{\Delta t^2}\cdot t^2\right\}$	$\exp\left\{-\frac{4\ln 2}{\Delta\tau}\cdot\tau^2\right\}$	$\sqrt{2}$	0.441
Squared hyperbolic secant	$\operatorname{sech}^2\left\{\frac{1.76t}{\Delta t}\right\}$	$3\left\{\left(\frac{2.72\tau}{\Delta\tau}\right)\cdot\coth\left(\frac{2.72\tau}{\Delta\tau}\right)-1\right\}\Big/\sinh^2\left(\frac{2.72\tau}{\Delta\tau}\right)$	1.55	0.315
Lorentzian	$1/\{1-4t^2/\Delta t^2\}$	$1/\{1-4\tau^2/\Delta\tau^2\}$	2	0.11
Single-sided exponential	$\exp\left\{-\frac{\ln 2}{\Delta t}\cdot t\right\} (t > 0)$	$\exp\left\{-\frac{2\ln 2}{\Delta\tau}\vert\tau\vert\right\}$	2	0.11

Δt and $\Delta\tau$ are the full widths at half maximum (FWHM) for the pulse $I(t)$ and the second-order autocorrelation $G^2(\tau)$, respectively.

$\Delta\nu$ is the FWHM of the measured frequency spectrum.

$\Delta\nu\,\Delta t$ is the time-bandwidth product of the pulse and its spectrum.

Appendix 2

The data acquisition program

The data acquisition program can be divided into six main categories:

1. Data Acquisition Project
   - Data Acquisition.c
   - Collection800.c
2. Communications
   - Serial Comm.c
   - LI.c
   - Actuator.c
   - NewBoxcar.c
3. NLS fitting
   - MickeyMousefit.NLS.c
   - funcs.c
   - covsrt.c
   - gaussj.c
   - mrqcof.c
   - mrqmin.c
   - nrutil_fixed.c
4. File management
   - Decode.c
   - file.mini.c
   - Utilities.c
   - Write.c
   - ReadSaveFile.c
5. Graphic Interface
   - mini.windows.c
   - dialogRoutines.c
   - plot.c
   - scale.c
   - Fit.Dlog.c
   - MousePaint.c
   - pleasewait.c
6. Header Files
   - Data Acquisition.h
   - DrInterface.h
   - GPIBres.h
   - IEEE.h
   - least_squares.h
   - NRUTIL.H
   - sys.h

Program listings follow (in alphabetical order); header files included at the end.

```
/*
        File Name:          Actuator.c
        Version:            1.0
        Project Name:       DataAcquisition
        Authors:            Mark J. Rosker
        Compiler:           Lightspeed C 3.01

        Utility:            This file contains EVERYTHING necessary for
                            actuator communication through function.one
                            of the RS-422 ports.  (Modem or Printer)

        Notes:              Keyboard commands are automatically capitalized.
*/

#include  "TextEdit.h"
#include  "Data Acquisition.h"

extern      PortConfig  actuatorPort;
extern      long        StartActCounts;

long  GetActPos(ActuatorNumber)     /* returns the actuator positon */
int         ActuatorNumber;
{
        Str255          AStr255;
        long        DecodePositionString();

        sprintf(AStr255, "%dTP\r" , ActuatorNumber);
        CtoPstr(AStr255);
        SendActStr255 (&AStr255, &actuatorPort);

        CountInHundreds(55);    /*    Has been carefully chosen! 45 to 50
                                                is minimum allowed!
        */
        GetReturnedStr(AStr255,&actuatorPort,FALSE,WAIT);

        return(DecodePositionString(&AStr255));
}

MoveActPos(ActuatorNumber, NewActPos)     /*    moves the actuator to
                                                the desired positon
        */
int         ActuatorNumber;
long  NewActPos;
{
        Str255          AStr255;

        sprintf(AStr255, "%dMA%0ld\r" , ActuatorNumber, NewActPos);
        CtoPstr(AStr255);
        SendActStr255 (&AStr255, &actuatorPort);

}

long  DecodePositionString(InStr255Ptr)/* Take str255 pointed to,
                                              decode into a long      */
Str255          *InStr255Ptr;
{
        char  StrPosition[20];
        char  *FirstCharPtr;
        int         i;
        long  atol();
        char  *strchr();

        for (i = 0; i < 20; i++) {
              StrPosition[i]  =  '\0';
        }

        PtoCstr(InStr255Ptr);
        FirstCharPtr = strchr(InStr255Ptr, ' ');/* find the space   */
        FirstCharPtr++;   /* point to first character past space    */

        strcpy(StrPosition,FirstCharPtr);/* copy the rest of string
                                              to StrPosition    */

        return(atol(StrPosition));
}


ActuatorInit()    /* Initializes the contorller and the actuators */
{
/* All commands are explained in the owners manual
                                         from Control Techniques */
        SendActStr255  ("\pEF\r",  &actuatorPort);
        sleep60(5);
        SendActStr255  ("\p1SU127\r",  &actuatorPort);
                         /* Sets velocity of actuator 1 to 127     */
        sleep60(5);
        SendActStr255  ("\p1SG62\r",  &actuatorPort);
                         /* Sets gain of actuator 1 to 62          */
        sleep60(5);
        SendActStr255  ("\p1MN\r",  &actuatorPort);
                         /*    Enables motion of actuator 1        */
        sleep60(5);
        SendActStr255  ("\p2SU127\r",  &actuatorPort);
                               /* Sets velocity of actuator 2 to 127
        */
        sleep60(5);
        SendActStr255  ("\p2SG62\r",  &actuatorPort);
                                     /* Sets gain of actuator 2 to 62
              */
        sleep60(5);
        SendActStr255  ("\p2MN\r",  &actuatorPort);
                                     /*    Enables motion of actuator 2
              */
}

int   SendActChar(theChar,  myPort)  /* to send each character     */
char        theChar;
PortConfig  *myPort;

{
        long                count;

        count = 1L;

        if (NOCOMMUNICATE) return;
```

```
                                        /* Capitalize theChar, if necessary */
      Capitalize(&theChar);

      if (FSWrite((*myPort).OutRefNum,&count,&theChar) != noErr) {
            DebugStr("\pFSWrite error");
      } else {
            /*DebugInt( (int) count);*/
      }


}

int   SendActStr255 (theStr255, myPort)   /* Send a string          */
Str255            theStr255;  /* Note: Assumes a Pascal string
      */
PortConfig  *myPort;

{
      long  count;
      char  *achar;
      int         i;

      if (NOCOMMUNICATE) return;

      achar = (char*)   theStr255;  /*    points to first character
                                                of Pascal string,
the length*/
      count = (long) *achar;  /*    gets the length of the string,
                                                as a long   */

      for (i = 1; i <= (int) count; i++) {
            Capitalize(achar + i);
                        /* capitalize the i-th character of string
      */
      }

      if (FSWrite((*myPort).OutRefNum,&count,achar+1) != noErr)
      {
            DebugStr("\pFSWrite error");
      }
}
```

```
/*
       File Name:          Collection.c
       Version:            1.0
       Project Name:       DataAcquisition
       Authors:            Mark J. Rosker
       Modifications:      Marcos Dantus and Lawrence W. Peng
       Compiler:           Lightspeed C 3.01

       Utility:            This file controls the data acquisition
                           procedure
                           See comments under the routine DoDataScan()

       Notes:              Modified slightly by LWP-principal change is in
                           how data files are saved
                           This version was modified for longer time scans
                           allowing for enough time for the actuators to
                           return to t = 0 (MD).
                           Statements were also included to safegard
                           against sending an actuator outside its travel
                           range (MD).
*/

#include "NRUTIL.H"
#include "Data Acquisition.h"

typedef struct {
       int                 Shots;
       int                 DataPoints;
       int                 Scans;
       long         ActCounts;
       long         StartCounts;
       int                 Device;
       Boolean             ActuatorOn;
       Boolean             Discriminate;
       int                 UpperDisPercent;
       int                 LowerDisPercent;
       Boolean             Normalize;
       int                 NormKind;
       int                 NormExponent;
} DataRunInfoStruct;

#define            LENGTHOFPARAMS          28L
#define            DIVIDENORMALIZE         1
#define            SUBTRACTNORMALIZE 2

extern   char filename[FILENAMELENGTH];
/*    file name of output-made extern by LWP*/

extern       PortConfig  actuatorPort;
extern       PortConfig  boxcarPort;
extern       float       *XData;
extern       float       **YData;
extern       int                NumYRows,  CurrYRow;
extern       int                NumOfShots,NumOfDataPts,NumOfScans;
extern       int                ScanCounter;
extern       long        ActCountsPerDP;
extern       long        StartActCounts;
extern       int                RunInProgress;
extern      int                 TotalNumOfShotsInRun;
extern      int                 CheckOnActPos;
extern      Boolean            DiscriminateOn;
extern      int                 UpperDisPercent;
extern      int                 LowerDisPercent;

extern      int                 SamplePts;  /* In discrim., the number of
                                                 sample pts to average*/

extern      Boolean             MoveTheActuator;
extern      Boolean             DataTakenFlag;

extern      int                 WhichNormalize;
extern      int                 NormalizePower;
extern      Boolean             NormalizeOn;

static      double              AvgOfY;
static      double              AvgOfY2;

extern      MenuHandle  myMenus[NUMOFMENUS];

extern      WindowPtr   PlotWindPtr;

extern      Str255              JunkStr255;
extern      char        JunkCharArray[255];

extern      int                 DataRunParameterType;
extern      Str255              ParamFileName[10];      /*increased to 10
by LWP

                            from 5*/

#define  MacTicker (*(long*)0x16A)
extern  Str255           JunkStr255;
static      DataRunInfoStruct CurrentParamInfo; /*made static by LWP*/

WindowPtr   TempPort;

int   DoDataCollectionMenu(item)
int         item;

{
      switch (item) {
      case dcParameters:
            DoParametersDlog();
            break;
      case dcStart:
            switch (RunInProgress) {
                  case NOTINPROGRESS:
                        InitADataRun();
                        break;
                  case INPROGRESS:
                  case INPAUSE:
                        RunOver();
                        break;

            }
            break;
```

```
        case  dcPause:
              switch  (RunInProgress)  {
                    case  INPROGRESS:
                          RunInProgress  =  INPAUSE;
                          SetItem
                          (myMenus[datacollectionM],
dcPause,"\pContinue");
                          break;
                    case  INPAUSE:
                          RunInProgress  =  INPROGRESS;
                          SetItem
                          (myMenus[datacollectionM],  dcPause,"\pPause");
                          break;
              }
              break;
        case  dcAmpAC:
        case  dcCPMAC:
        case  dcICN:
        case  dcMixedSpectrum:
        case  dcSHGSpectrum:
        case  dcNal:
        case  dcXCorr:
              CheckItem
              (myMenus[datacollectionM],  DataRunParameterType,  FALSE);
              LoadParams(ParamFileName[item  -  dcAmpAC]);
              CheckItem  (myMenus[datacollectionM],  item,  TRUE);
              DataRunParameterType  =  item;
              DoParametersDlog();
              break;
        case  dcSaveParams:
              CurrentParamInfo.Shots                 =      NumOfShots;
              CurrentParamInfo.DataPoints     =     NumOfDataPts;
              CurrentParamInfo.Scans                 =      NumOfScans;
              CurrentParamInfo.ActCounts             =      ActCountsPerDP;
              CurrentParamInfo.StartCounts    =     StartActCounts;
              CurrentParamInfo.Device                =
        actuatorPort.whichDev;
              CurrentParamInfo.ActuatorOn            =      MoveTheActuator;
              CurrentParamInfo.Discriminate =       DiscriminateOn;
              CurrentParamInfo.UpperDisPercent=     UpperDisPercent;
              CurrentParamInfo.LowerDisPercent=     LowerDisPercent;
              CurrentParamInfo.Normalize             =      NormalizeOn;
              CurrentParamInfo.NormKind              =      WhichNormalize;
              CurrentParamInfo.NormExponent =       NormalizePower;

              SaveParams(ParamFileName[DataRunParameterType  -  dcAmpAC],
                                         &CurrentParamInfo,
LENGTHOFPARAMS  );
              break;
        case   dcSaveCricketGraph:
        /*added  LWP*/
              WriteData(XData,YData,NumOfDataPts,NumYRows,CurrYRow);
              break;
        default:                                /*added  LWP*/
              break;
        }
}
```

```
SetUpParamFiles()
/* This routine must have the correct adress for each param file */
{
	strcpy(ParamFileName[dcAmpAC-dcAmpAC],
		"\pFemtosec HD:System Folder:DAParams:Amp AC Params");

	strcpy(ParamFileName[dcCPMAC-dcAmpAC],
		"\pFemtosec HD:System Folder:DAParams:CPM AC Params");

	strcpy(ParamFileName[dcICN-dcAmpAC],
		"\pFemtosec HD:System Folder:DAParams:ICN Params");

	strcpy(ParamFileName[dcMixedSpectrum-dcAmpAC],
		"\pFemtosec HD:System Folder:DAParams:Mix Sp Params");

	strcpy(ParamFileName[dcSHGSpectrum-dcAmpAC],
		"\pFemtosec HD:System Folder:DAParams:SHG Sp Params");

	strcpy(ParamFileName[dcNaI-dcAmpAC],
		"\pFemtosec HD:System Folder:DAParams:NaI Params");

	strcpy(ParamFileName[dcXCorr-dcAmpAC],
		"\pFemtosec HD:System Folder:DAParams:XCorr Params");

	LoadParams(ParamFileName[0]);
	CheckItem (myMenus[datacollectionM], dcAmpAC, TRUE);
	DataRunParameterType = dcAmpAC;
}

DoParametersDlog()
{
	DialogPtr	theDialog;
	int			itemHit;
	WindowPtr	tmpWindPtr;
	extern		DialogRecord	DlogRec;
	Str255		theText;
	Handle		theTextHdl;
	int			theType;
	Rect		txtBox;

	int			atoi();

	int			TempNumOfShots		= NumOfShots;
	int			TempNumOfDataPts	= NumOfDataPts;
	int			TempNumOfScans		= NumOfScans;
	long		TempActCountsPerDP	= ActCountsPerDP;
	long		TempStartActCounts	= StartActCounts;
	Boolean		TempMoveTheActuator	= MoveTheActuator;
	int			TempWhichAct		= actuatorPort.whichDev;
	Boolean		TempDiscriminateOn	= DiscriminateOn;
	int			TempUpperDisPercent	= UpperDisPercent;
	int			TempLowerDisPercent	= LowerDisPercent;
	Boolean		TempNormalizeOn		= NormalizeOn;
	int			TempWhichNormalize	= WhichNormalize;
	int			TempNormalizePower	= NormalizePower;

	if (TempNumOfDataPts <= 0)
		TempNumOfDataPts = 1;
```

```
if (TempWhichAct != 1 && TempWhichAct != 2)
      TempWhichAct = 1;

if (TempWhichNormalize != 1 && TempWhichNormalize != 2)
      TempWhichNormalize = 1;

GetPort(&tmpWindPtr);          /* Point at current window    */

theDialog = GetNewDialog(PARAMETERSDLOGID, &DlogRec,
            (WindowPtr)(-1) );
SetPort(theDialog);

SetUpTEDlogInt(theDialog,6,TempNumOfDataPts);
SetUpTEDlogInt(theDialog,8,TempNumOfShots);
SetUpTEDlogLInt(theDialog,11,TempActCountsPerDP);
SetUpTEDlogLInt(theDialog,13,TempStartActCounts);
SetUpTEDlogInt(theDialog,4,TempNumOfScans);
ItemCheckMark(theDialog, 16, TempMoveTheActuator);
ItemCheckMark(theDialog, 19, TempDiscriminateOn);
ItemCheckMark(theDialog, 24, TempNormalizeOn);

if (TempMoveTheActuator)
      ItemCheckMark(theDialog, 16 + TempWhichAct, 1);
                                    /* TempWhichAct = 1 or 2
*/
else {
      HideDItem(theDialog,17);
      HideDItem(theDialog,18);
}

if (TempDiscriminateOn) {
      SetUpTEDlogInt(theDialog,21,TempUpperDisPercent);
      SetUpTEDlogInt(theDialog,23,TempLowerDisPercent);
} else {
      HideDItem(theDialog,20);
      HideDItem(theDialog,21);
      HideDItem(theDialog,22);
      HideDItem(theDialog,23);
}

if (TempNormalizeOn) {
      ItemCheckMark(theDialog, 24 + TempWhichNormalize, 1);
                              /* TempWhichNormalize = 1 or 2
*/
      SetUpTEDlogInt(theDialog,27,TempNormalizePower);
} else {
      HideDItem(theDialog,25);
      HideDItem(theDialog,26);
      HideDItem(theDialog,27);
}

ShowWindow(theDialog);
frameOkbox(theDialog);

do{
      ModalDialog(NIL, &itemHit);
```

```
                                        /*calls  filter  before
handling*/
           switch  (itemHit){
                case 1 :                /* Ok  button
     */
                case 2 :                /* Cancel  button       */
                     break;

                case 4 :                /* Number  of  scans
     */
                     DoTEDlogInt(theDialog,itemHit,&TempNumOfScans);
                     break;
                case 6 :                /* Number  of  data  points*/

     DoTEDlogInt(theDialog,itemHit,&TempNumOfDataPts);
                     break;
                case 8 :           /* Number  of  shots/data  pt.   */
                     DoTEDlogInt(theDialog,itemHit,&TempNumOfShots);
                     break;
                case 11 :          /* Number  of  act  cts/data  pt.*/

     DoTEDlogLInt(theDialog,itemHit,&TempActCountsPerDP);
                     break;
                case 13 :          /* Start  actuator  position    */

     DoTEDlogLInt(theDialog,itemHit,&TempStartActCounts);
                     break;
                case 16 :   /* Toggle  moving  the  actuator  (Check
box)*/
                     if  (TempMoveTheActuator)  {
                          ItemCheckMark(theDialog,  16,  0);
                               /* uncheck  'move  actuator'  box
     */

                          HideDItem(theDialog,17);
                          HideDItem(theDialog,18);
                          TempMoveTheActuator  =  FALSE;
                     } else {
                          ItemCheckMark(theDialog,  16,  1);
                               /* check  'move  actuator'  box  */
                          ShowDItem(theDialog,17);
                          ShowDItem(theDialog,18);
                     ItemCheckMark(theDialog,  16  +  TempWhichAct,  1);
                          TempMoveTheActuator  =  TRUE;
                     }
                     break;

                case 17:           /* radio  button  for  actuator  *1
     */
                     ItemCheckMark(theDialog,  16  +  TempWhichAct,  0);
                     TempWhichAct  =  1;
                     ItemCheckMark(theDialog,  16  +  TempWhichAct,  1);
                     break;

                case 18:           /* radio  button  for  actuator  *2
     */
                     ItemCheckMark(theDialog,  16  +  TempWhichAct,  0);
                     TempWhichAct  =  2;
                     ItemCheckMark(theDialog,  16  +  TempWhichAct,  1);
```

```
                        break;
                case 19 :   /* Toggle discriminate on (Check box)
        */
                        if (TempDiscriminateOn) {
                            HideDItem(theDialog,20);
                            HideDItem(theDialog,21);
                            HideDItem(theDialog,22);
                            HideDItem(theDialog,23);
                            ItemCheckMark(theDialog, itemHit, 0);
                            TempDiscriminateOn = FALSE;
                        } else {
                            ShowDItem(theDialog,20);
                            ShowDItem(theDialog,21);
                            ShowDItem(theDialog,22);
                            ShowDItem(theDialog,23);
                            SetUpTEDlogInt
                                (theDialog,21,TempUpperDisPercent);
                            SetUpTEDlogInt
                                (theDialog,23,TempLowerDisPercent);
                            ItemCheckMark(theDialog, itemHit, 1);
                            TempDiscriminateOn = TRUE;
                        }
                        break;
                case 21 :                           /* Number of */

        DoTEDlogInt(theDialog,itemHit,&TempUpperDisPercent);
                        break;
                case 23 :                           /* Number   */

        DoTEDlogInt(theDialog,itemHit,&TempLowerDisPercent);
                        break;

                case 24 :   /* Toggle discriminate on (Check box)
        */
                        if (TempNormalizeOn) {
                            HideDItem(theDialog,25);
                            HideDItem(theDialog,26);
                            HideDItem(theDialog,27);
                            ItemCheckMark(theDialog, itemHit, 0);
                            TempNormalizeOn = FALSE;
                        } else {
                            ShowDItem(theDialog,25);
                            ShowDItem(theDialog,26);
                            ShowDItem(theDialog,27);

        SetUpTEDlogInt(theDialog,27,TempNormalizePower);
                            ItemCheckMark(theDialog, itemHit, 1);
                            ItemCheckMark
                            (theDialog, 24 + TempWhichNormalize, 1);

                                                    /*
    TempWhichNormalize = 1 or 2*/
                            TempNormalizeOn = TRUE;
                        }
                        break;

                case 25:
```

```
                /* radio button for normalization*1 y = ADC1/ADC2**n
*/
                        ItemCheckMark
                                (theDialog, 24 + TempWhichNormalize, 0);
                        TempWhichNormalize = DIVIDENORMALIZE;
                        ItemCheckMark
                                (theDialog, 24 + TempWhichNormalize, 1);
                        break;

                case 26:/* radio button for normalization *1
                                                y = ADC1 - ADC2  */
                        ItemCheckMar
                                k(theDialog, 24 + TempWhichNormalize, 0);
                        TempWhichNormalize = SUBTRACTNORMALIZE;
                        ItemCheckMark
                                (theDialog, 24 + TempWhichNormalize, 1);
                        break;
                case 27 :         /* Number of shots/data pt.   */

        DoTEDlogInt(theDialog,itemHit,&TempNormalizePower);
                        break;
                default:
                        break;

                }                       /*the case of what item was hit*/
        }while (itemHit != 1 && itemHit != 2);

        CloseDialog(theDialog);
        SetPort(tmpWindPtr);

        if (itemHit != 2){      /* Unless it was cancel...          */
                MoveTheActuator                 =       TempMoveTheActuator;
                NumOfShots                      =       TempNumOfShots;
                NumOfDataPts                    =       TempNumOfDataPts;
                NumOfScans                      =       TempNumOfScans;
                ActCountsPerDP                  =       TempActCountsPerDP;
                StartActCounts                  =       TempStartActCounts;
                actuatorPort.whichDev   =       TempWhichAct;
                DiscriminateOn                  =       TempDiscriminateOn;
                UpperDisPercent                 =       TempUpperDisPercent;
                LowerDisPercent                 =       TempLowerDisPercent;
                NormalizeOn                     =       TempNormalizeOn;
                WhichNormalize                  =       TempWhichNormalize;
                NormalizePower                  =       TempNormalizePower;
        }
}


RunOver()               /* call when all done with a complete run*/
{
        float fitprogram();

        SysBeep(5);             /* signal all done      */
        WriteData(XData,YData,NumOfDataPts,NumYRows,CurrYRow);

        RunInProgress = NOTINPROGRESS;
        DisableItem( myMenus[datacollectionM], dcPause );
        EnableItem( myMenus[datacollectionM], dcStart  );
```

```
	SetItem  (myMenus[datacollectionM],  dcStart,"\pStart");
	ScanCounter = 0;
	if  (MoveTheActuator)    /* special return to start position */
		MoveActPos(actuatorPort.whichDev,  StartActCounts);
	SetCursor(&arrow);
}

InitADataRun()    /* Set-up all parameters for data collection    */
{
	int             i;
	float       GetBoxcarVoltage();

	MakeDataArrays();

	ScanCounter         =      0;

	if  (MoveTheActuator)/* special  return  to  start  position
	                                        -- should not be needed
	*/
		MoveActPos(actuatorPort.whichDev,  StartActCounts);

	for (i = 0; i < NumOfDataPts ; i++) {
		XData[i]= (float) ((i * ActCountsPerDP) + StartActCounts);
	}
	TotalNumOfShotsInRun = 0;

	EnableItem( myMenus[datacollectionM], dcPause );
	SetItem  (myMenus[datacollectionM], dcStart,"\pStop");

	sprintf(  filename,"%s","");
	                    /*clear out the old filename-added by LWP*/
	sprintf(  filename,"%s","Most  Current  Scan");
	                                    /*name the new file-added by LWP*/
	CreateNewFile(filename);                /*clear new file-added by
LWP*/

	RunInProgress = INPROGRESS;

	DrawPlotWind(FALSE,XData,YData,NumOfDataPts,
	          NumYRows,CurrYRow,NIL);/*   Do plot (sets up axes)
	*/

	DataTakenFlag = TRUE;   /* once made true, always stays so  */
	PleaseWait();

}

FreeDataArrays()
{
	free_vector(XData,0,NumOfDataPts  -1  );
	free_matrix(YData,0,NumYRows - 1, 0,NumOfDataPts -1);
	XData = YData = NIL;
	/*    NumOfDataPts = NumYRows = CurrYRow = 0;    */
}

MakeDataArrays()
{
	register int i,j;
```

```
        if  (XData  !=  NIL)  FreeDataArrays();
        XData  =  vector(0,NumOfDataPts  -1  );
        YData  =  matrix(0,NumYRows  -  1,  0,NumOfDataPts  -1);

        for  (i  =  0;  i  <  NumOfDataPts;  i++)  {
                XData[i]  =  0.0;
                for  (j  =  0;  j  <  NumYRows;  j++)  {
                        YData[j][i]  =  0.0;
                }
        }
}

DoDataScan()
/*
 Routine  to  take  one  complete  scan,
 and  then  return  main  event  loop.   Note  that  this  will  not  take  us
 back  to  the  main  event  loop  at  an  adequate  rate;  at  some  point  I
 expect  to  make  DPCounter  an  external  variable,  remove  the  FOR  loop
 over  DPCounter,  and  call  this  routine  only  once  per  main  event  loop.
 Now  iterates  ScanCounter.

 Please  read  this  before  making  modifications:
 After  a  great  deal  of  experiment,  I  [MJR]  have  concluded  that  the
 actuator  behavior  at  20  Hz  is  as  follows:
 (1)  It  takes  on  the  order  of  70  ms  to  execute  the  loop:
        do  {
                TargetError  =  labs(GetActPos(actuatorPort.whichDev)

                        -  Target);
                }  while  (  TargetError   >  MAXACTERROR  )  ;
        This  is  too  slow  to  keep  up  at  20  Hz.   The  70  ms  number  seems
        fairly  independent  of  the  number  of  steps  travelled  (in  the
        range  of  5  -  50  steps),  so  is  likely  due  to  the  fixed  start  and
        stop  times.

 (2)  Several  modes  are  allowed,  depending  on  the  variable
        CheckOnActPos.   These  are:
        (a)    CheckOnActPos  =  CHECKONLYATZERO  .   Guaranteed  to  work  at  20
                Hz,  but  dangerous.  Although  hard  to  prove,  for  a  setting  of
                ActCountsPerDP  =  30,  I  found  the  average  error  immediately
                after  a  boxcar  read  was  18  counts!   Note  that  this  option,
                like  all  options,  does  verify  the  actuator  is  at
                StartActCounts  (i.e.,  sends  a  "TP")  for  the  first  data
point.
        (b)    CheckOnActPos  =  CHECKNOLOOP  .   This  mode  does  a  "TP"  AFTER
                each  data  point.If  the  TargetError  is  >  MAXACTERROR  (which
                should  be  set  to  be  quite  large)  the  data  collection  is
                interrupted  and  a  dialog  box  put  up.  The  accumulated
                Target  errors  are  kept  track  of.
        (c)  CheckOnActPos  =  CHECKLOOP  .   This  does  a  "TP"  BEFORE  taking
                data.   Data  taking  is  held  up  until  the  Actutator  is  within
                MAXACTERROR  of  its  target.   Typically,  this  means  a  limit  of
                about  10  -  15  Hz  maximum  collection  rate.   This  is  the
                most  conservative  mode,  since  the  actuator  must  be  very
                nearly  where  you  want  it  before  each  data  point.

 (3)  My  best  recommendation  for  data  collection  at  present:
        Use CHECKNOLOOP to maximize collection efficiency, but set
        NumOfShots = 2.  This will mean the first shot will be off
        somewhat, but the second should be OK.  More importantly, this
        should give the actuator a chance to keep up.

   (4) A final point: I have tried to put timing loops (e.g., count to
        some large number) so that the actuator has more time to get to
        desired position before taking a boxcar data point.  As might be
        exected, this really doesn't help.  My conclusion is that timing
        loops can only help in unusual situations, and then only
        slightly, so they should be avoided.

   (5) NOTE: The timing is SO tight that I have seen CHECKNOLOOP
        sometimes work (get every trigger), sometimes not.  Obviously,
        if it doesn't get every trigger it's better to instead use
        CHECKLOOP.  This gets every other trigger almost always.

note:   the order of activity is as follows:
        (1) Assume the actuator has already been moved to the right place
        (2)  Check to make sure it's there -- loop 'til it is
        (3)  Take the data points from boxcar (NumOfShots data points)
        (4) Send the actuator command to move to the next point

ADDITIONAL NOTE ADDED BY LWP (NOV 1989):  The collection routine now
saves the most currently averaged file immediately after the plot
window is updated. The previously saved averaged file will be
overridden by the new one.If one want to save (as an example) every
10 files, then you need an internal counter to keep track of the
scan number.  You will also need to create another file name in
which to store the data.

ADDITIONAL NOTE ADDED BY MD (DEC 1989):  The program instead of
having a static MAXACTERROR it will have a maximum error of 10% for
each position.  Since we always take three points per position these
will be adecuate.  Also this is very important when taking data
with very small steps such as 5 steps or smaller */
{
        long  GetActPos(), labs();
        float GetBoxcarVoltage();
        float MakeFakeData();
        long  TargetError,TargetErrorSum,Target;
        float LatestPointAvg, VoltageSum, SignalVoltage, RefVoltage,

                              ratio;
        int           ShotCounter,DPCounter;
        int           i, j, iter;
        Point APoint;
        long  ALong;
        double        GetMean(), RefMean;

        int           BadCounter;
        static        long          GlobalAcceptCounter;
        static        long          GlobalTriggerCounter;
        int           MaxActError;

        /*register  long  beginticks,endticks;*/

        WindowPtr   TempPort;
        GetPort(&TempPort);             /*     Save  the  current  port
        */
        SetPort(PlotWindPtr);           /*     Point  to  plot  window
        */

        /* Note: all plotting done in this routine is not in the picture!
           It will not be updated if an update event happens.  At present,
           I can live with that  */

        MoveTo(55,25);
        sprintf(JunkStr255,"Current  scan:  %d",ScanCounter  +  1);
        CtoPstr(JunkStr255);
        ForeColor(cyanColor);
        DrawString(JunkStr255);
        ForeColor(blueColor);


        /* if Discrimination is on, get statistics on first SamplePts

data  points */
        if  (!NOCOMMUNICATE && DiscriminateOn && (ScanCounter == 0))  {
              GlobalTriggerCounter  =  0;
              GlobalAcceptCounter  =  0;
              for  (i = 0, AvgOfY = 0.0, AvgOfY2 = 0.0;  i < SamplePts;  i++)
{
                    RefVoltage  =  GetBoxcarVoltage(2);
                    UpdateYSums(i,RefVoltage);
              }
        }
        /*printf("Mean  =  %g,  StdDev  =  %g\n",GetMean(SamplePts),

        GetStdDev(SamplePts));*/

        if  (DataRunParameterType  ==  dcMixedSpectrum  ||
                                  DataRunParameterType  ==  dcSHGSpectrum)
              do  {
                    ;
              } while  (GetBoxcarVoltage(3)  >  2.0);


        MaxActError  =  (XData[1]  -  XData[0])/10;
        if  (MaxActError  ==  0)  MaxActError  =  1;
                          /* Make MaxActError 10% of the step size or 1 */

ABORT:
        for  (DPCounter  =  0,  TargetErrorSum  =  0;
                                        DPCounter  <  NumOfDataPts;
DPCounter++)  {

        if (Button())  {   /*   Emergency Stop!    */
              DPCounter--;
              DebugString("CurrentScan  Aborted  by  User    g  to  rescan,
                          ea  to  restart  program");
              goto  ABORT;
              }
```

```
            Target = (long) XData[NumOfDataPts -1];   /* get max x
value*/
                        if (Target > 400000) {
                              ParamText("\pActuator will not go past
400000
                                    (DoubleClick to
Abort)","\p","\p","\p");
                              SysBeep(4);
                              CautionAlert(GeneralAlert,NIL);
                        }

            Target = (long) XData[DPCounter];   /* get next x value
      */



            if (!NOCOMMUNICATE && MoveTheActuator &&
                                    /* Do I loop over actuator pos.?
      */
                  (DPCounter == 0 || CheckOnActPos == CHECKLOOP)) {

                  iter = 0;
                  do {
                  /* loop until the actuator gets where it's supposed
                                                      to be      */
                        CountInHundreds(20);

                        if (iter == 800) {            /* time to just
give up!*/
                              ParamText
                        ("\pThe actuator is not
responding","\p","\p","\p");
                              SysBeep(4);
                              CautionAlert(GeneralAlert,NIL);
                        }

                        if (++iter % 10 == 0) {
                        /* special re-send move command     */
                              MoveActPos(actuatorPort.whichDev,Target);
                        }

                        TargetError =
labs(GetActPos(actuatorPort.whichDev)
                                    - Target);
                        TargetErrorSum = TargetErrorSum +   TargetError;
                  } while ( TargetError   > MaxActError );

            /*DebugInt(iter);*/
            }
            /* end if actuator is supposed to loop til it gets to the
             right place     */

   /* take the data       */
            for (i = 1, VoltageSum = 0.0; i <= NumOfShots; i++) {
REACQUIRE:
                  if (!NOCOMMUNICATE) {
                        /* get the data....     */
                        if (DiscriminateOn) {
                                RefMean  =  GetMean(SamplePts);
                                BadCounter  =  0;

                                do {
                                     if (++BadCounter % 10 == 0)
                                          SysBeep(1);
                                     GetBoxcarVoltage2
                                     (1,2,&SignalVoltage,  &RefVoltage);
                                     UpdateYSums(SamplePts,RefVoltage);
                                     GlobalTriggerCounter++;
                                } while ( (RefVoltage / RefMean) >
                                     (1.0  +  UpperDisPercent/100.0)   ||
                                          (RefVoltage / RefMean) <
                                          (1.0  -  LowerDisPercent/100.0)
);

                                GlobalAcceptCounter++;

                           } else if (NormalizeOn)
                           GetBoxcarVoltage2(1,2,&SignalVoltage,
&RefVoltage);
                           else
                                SignalVoltage  =  GetBoxcarVoltage(1);
                                /* take real data */

                           /* now process the data...     */
                           if (NormalizeOn) {
                                if (WhichNormalize == DIVIDENORMALIZE) {
                                     for (j = NormalizePower; j > 0; j--)
                                          if (RefVoltage == 0.0)
                                          RefVoltage  =  0.0000001;
                                          /*protect from divide by zero
*/

                                          SignalVoltage /= RefVoltage;
                                          if(fabs(SignalVoltage) > 20.0)
                                               goto REACQUIRE;
                                     VoltageSum += SignalVoltage;
                                } else if (WhichNormalize ==
SUBTRACTNORMALIZE)
                                VoltageSum += (SignalVoltage -
RefVoltage);
                                else
                                     DebugStr("\pBad value of
NormalizeOn");
                           } else
                                VoltageSum += SignalVoltage;


                      } else {    /* make up data          */
                           VoltageSum =  VoltageSum + MakeFakeData(Target);
                      }

                      if (!NOCOMMUNICATE && MoveTheActuator &&
                                CheckOnActPos == CHECKNOLOOP) {
                           TargetError =
labs(GetActPos(actuatorPort.whichDev)
                                 - Target);
                           TargetErrorSum = TargetErrorSum +  TargetError;
                           if ( TargetError  > MAXACTERROR ) {
                                sprintf  (JunkCharArray,
                                "Actuator out of position by %ld counts",
                                     TargetError);
                                CtoPstr(JunkCharArray);
                                ParamText(JunkCharArray,"\p","\p","\p");
                                SysBeep(4);
                                CautionAlert(GeneralAlert,NIL);
                           }
                      }
                 }          /* end for loop over NumOfShots          */

  /* move the actuator to its next destination */
                 if (!NOCOMMUNICATE && MoveTheActuator) {
                                     /* move to next point right away
      */
                      MoveActPos(actuatorPort.whichDev,
                      (long)  (XData[(DPCounter+1)  %  NumOfDataPts]));
                                     /* % sends DPCounter+1 -> 0   */
                 }


/* enter data into arrays */
                 LatestPointAvg  =  VoltageSum/(float)NumOfShots;
                 /* this is the average value of latest data     */
                 YData[CurrYRow][DPCounter]
                      =  (  (YData[CurrYRow][DPCounter]  *
TotalNumOfShotsInRun)
                           + VoltageSum  ) / (TotalNumOfShotsInRun  +
NumOfShots);
      /*DP[DPCounter].NumAvgs  =  DP[DPCounter].NumAvgs  +  NumOfShots;*/


  /* Draw the point (remember, this is not in the picture, so no
                                                updates)    */
                 DrawADataPoint(  XData[DPCounter],  LatestPointAvg);

      }           /* end loop over DPCounter -- done with the scan */


/*    endticks  =  MacTicker;
      DebugLInt(endticks  -  begintickes);*/

      ScanCounter++;
      TotalNumOfShotsInRun  =  TotalNumOfShotsInRun  +  NumOfShots;

      ForeColor(blackColor);
      DrawPlotWind(TRUE,XData,YData,NumOfDataPts,NumYRows,CurrYRow,
           NIL);       /*  Do plot (axes and data)              */

      sprintf(  filename,"%s","");
                                /*clear out the old filename-added by
LWP*/
      sprintf(  filename,"%s","Most  Current  Scan");
                                     /*name the new file-added by LWP*/
      SaveNewFile(filename,NumOfDataPts,0);
                                     /*Save averaged file--added LWP*/

      /* Let the monochromator go back to its next destination */
```

```
            if (DataRunParameterType == dcMixedSpectrum ||
                    DataRunParameterType == dcSHGSpectrum)
                  do {
                  ;
            } while (GetBoxcarVoltage(3) < 2.0);

      /* if Discrimination is on, get statistics on first SamplePts
            data points      */
      if (!NOCOMMUNICATE && DiscriminateOn) {
            MoveTo(500,25);
            ratio = ( 100.0 * (float) GlobalAcceptCounter) /
                  ( (float) GlobalTriggerCounter);
            sprintf(JunkStr255,"Accept:  %.1f  %%",ratio);
            CtoPstr(JunkStr255);
            ForeColor(cyanColor);
            DrawString(JunkStr255);
            ForeColor(blueColor);
      }
      SetPort(TempPort);            /*    Point to original window
      */
}

float MakeFakeData(x)
long  x;
{
      int   i;
      unsigned int seed;

/*    GetDateTime(&randSeed);
      for (i = 0; i < FakeNumOfDataPts; i++) {
            FakeDP[i].DataX = (float) i;
            FakeDP[i].DataY = ((float) i) +
                  (((float) Random() + 32767.0) / 100.0);
      }*/

      return(( ( (float) Random())/32767.) +
                              (0.8 * sin(2.0 * (float) x /128.)));

}

SaveParams( fn, APtr,count )
                        /* fn assumed to be a Pascal string!
      */
Str255      *fn;
Ptr         APtr;
long  count;
{
int         *vRef;
      int   refNum;
      long  countIn;

      countIn = count;

      *vRef = 0;
      if (CreateParamFile(fn, vRef, &refNum)) {
            FSWrite(refNum, &count, APtr);
            if (count != countIn) FileError
```

```
                              ("\pError writing parameter file ", fn);
                FSClose( refNum );
                return(1);
        }
        else {
                DebugStr("\p Why  am i here?");
                FileError("\pError creating parameter file ", fn);
        }
        return(0);
}

CreateParamFile( fn, vRef, theRef )
Str255          *fn;
int             *vRef;
int             *theRef;
{
        int io;

        io=Create(fn, *vRef, 'DTAQ', 'PARM');
        if ((io==noErr) || (io==dupFNErr))
                                        io = FSOpen( fn, *vRef, theRef );
        return( (io==noErr) || (io==dupFNErr) );
}

ReadParamFile( fn, APtr, count )
Str255          fn;
Ptr             APtr;
long    count;
{
        int     refNum;
        int             io;
        int             vRef;

        vRef = 0;

        if (FSOpen( fn, vRef, &refNum)==noErr) {
                if ( (io =  FSRead( refNum, &count, APtr )) != noErr) { ;
                        FileError( "\pError reading parameter file: ", fn );
                }

                FSClose( refNum );
        }
        else FileError( "\pError opening parameter file: ", fn );
}

LoadParams(PascalFileName)
Str255          PascalFileName;                 /* in pascal, you moron!
        */
{

        ReadParamFile(PascalFileName,&CurrentParamInfo, LENGTHOFPARAMS);

        NumOfShots                      =       CurrentParamInfo.Shots;
        NumOfDataPts                    =       CurrentParamInfo.DataPoints;
        NumOfScans                      =       CurrentParamInfo.Scans;
        ActCountsPerDP                  =       CurrentParamInfo.ActCounts;
        StartActCounts                  =       CurrentParamInfo.StartCounts;
        actuatorPort.whichDev   =       CurrentParamInfo.Device;
```

```
	MoveTheActuator			=	CurrentParamInfo.ActuatorOn;
	DiscriminateOn			=	CurrentParamInfo.Discriminate;
	UpperDisPercent			=
	CurrentParamInfo.UpperDisPercent;
	LowerDisPercent			=
	CurrentParamInfo.LowerDisPercent;
	NormalizeOn			=	CurrentParamInfo.Normalize;
	WhichNormalize			=	CurrentParamInfo.NormKind;
	NormalizePower			=	CurrentParamInfo.NormExponent;
}

UpdateYSums(NumPts,LatestY)
int			NumPts;
double			LatestY;
{

	AvgOfY		= ((AvgOfY * NumPts)   + LatestY) / (NumPts + 1.0);
	AvgOfY2		= ((AvgOfY2 * NumPts) + (LatestY * LatestY)) /
		(NumPts + 1.0);

/*	AvgOfY		+= LatestY;
	AvgOfY2		+= (LatestY * LatestY);*/
}

double		GetStdDev(NumPts)
int		NumPts;
{
	double		value;

	value = sqrt(AvgOfY2 - AvgOfY*AvgOfY);

	return(value);
}

double		GetMean(NumPts)
int		NumPts;
{
	double		value;

	value = AvgOfY;

	return(value);
}
```

```
/*
      File Name:         covsrt.c
      Version:           1.0
      Project Name:      DataAcquisition
      Authors:           subroutine from Numerical Recipes
      Compiler:          Lightspeed C 3.01

      Utility:           This file calculates the covariance and
                         is used for NLS fitting
*/

void covsrt(covar,ma,lista,mfit)
float **covar;
int ma,lista[],mfit;
{
      int i,j;
      float swap;

      for (j=1;j<ma;j++)
            for (i=j+1;i<=ma;i++) covar[i][j]=0.0;
      for (i=1;i<mfit;i++)
            for (j=i+1;j<=mfit;j++) {
                  if (lista[j] > lista[i])
                        covar[lista[j]][lista[i]]=covar[i][j];
                  else
                        covar[lista[i]][lista[j]]=covar[i][j];
            }
      swap=covar[1][1];
      for (j=1;j<=ma;j++) {
            covar[1][j]=covar[j][j];
            covar[j][j]=0.0;
      }
      covar[lista[1]][lista[1]]=swap;
      for (j=2;j<=mfit;j++) covar[lista[j]][lista[j]]=covar[1][j];
      for (j=2;j<=ma;j++)
            for (i=1;i<=j-1;i++) covar[i][j]=covar[j][i];
}
```

```
/*
        File Name:          Data Acquisiton.c
        Version:            1.0   (DA v2 , 10/22/88)
        Project Name:       DataAcquisition
        Authors:            Graphic interphase by LightSpeedC for MiniEdit
                            project
                            Addapted by Mark J. Rosker
        Modifications:      Lawrence W. Peng
        Compiler:           Lightspeed C 3.01

        Utility:            This is the main() of the DataAcquisition
project
                            It controls the events and the graphic interface

        Notes:

        This program was adapted from Lightspeed C's 'Buggy edit'

        The sample application from Inside Macintosh (RoadMap p.15-17)
        beefed up a bit by Stephen Z. Stein, Think Technologies Inc.

        The resources used in this program are on disk LS3.Utilities
        in the file "new project.rsrc".  This resource file was created
        using ResEdit, so there is no corresponding RMaker source file.

        This version is  a modification of MJR's DA v1 (for use in 047)
        in an attempt to get the NLS routines working properly. LWP
*/

#include  "Data Acquisition.h"
#include  "least_squares.h"

int   CountEm = 0;

/*Window stuff*/
WindowRecord        CommWindRec,EditWindRec,PlotWindRec;
WindowPtr           CommWindPtr,EditWindPtr,PlotWindPtr;
PicHandle           PlotWindPic = NIL;

/*cursor stuff*/
CursHandle          ClockCursor;        /*handle to the waiting watch
cursor*/
CursHandle          TextCursor;         /*handle to the text entry cursor*/
CursHandle          CrossCursor;

Cursor                      editCursor;         /* old cursors stuff    */
Cursor                      waitCursor;

/*plot stuff*/
Rect                PlotAxesRect;

TEHandle            TEH;
int                         linesInFolder;
Rect                dragRect = { 0, 0, 1024, 1024 };
ControlHandle       vScroll;
char                dirty;

extern      Str255       theFileName;
```

```
MenuHandle          myMenus[NUMOFMENUS];

DialogRecord        DlogRec;            /* All the following added by MJR
     */
Str255                   JunkStr255;
Handle                   JunkHandle;
int                      JunkInt;
Rect                JunkRect;
char                JunkCharArray[255];

char                actuatorInBuff[256];
char                boxcarInBuff[256];
PortConfig          actuatorPort;
PortConfig          boxcarPort;
int                      deviceSend  =  DEVICESETACTUATOR;

float               *XData                 =      NIL;
float               **YData                =      NIL;
int                      NumYRows          =      1;
int                      CurrYRow          =      0;
int                      NumOfDataPts      =      256;
char                filename[FILENAMELENGTH];
                         /* file name of output-made extern by LWP*/

int                      NumOfShots        =      1;
int                      NumOfScans        =      1;
long                ActCountsPerDP    =     30L;
long                StartActCounts    =     0L;
Boolean                  MoveTheActuator   =      TRUE;
int                      ScanCounter       =      0;
int                      RunInProgress     =      NOTINPROGRESS;
Boolean                  DataTakenFlag     =      FALSE;
                                   /* is there anything to plot yet*/

int                      CheckOnActPos     =      CHECKLOOP;
                                          /* check act. position after
taking data*/
int                      TotalNumOfShotsInRun;
                                   /* number of shots in current run*/
Boolean             DiscriminateOn    =     TRUE;
int                      UpperDisPercent   =      25;
int                      LowerDisPercent   =      25;
int                      SamplePts         =      100;
                    /* In discrim., the number of sample pts to average
over*/

int                      WhichNormalize    =      1;
int                      NormalizePower    =      1;
Boolean                  NormalizeOn       =  TRUE;

/* Parameter file stuff */
int                      DataRunParameterType;
Str255                   ParamFileName[10];       /*increased to 10 by LWP
from 5*/

Boolean             FirstFit    = TRUE;
Boolean                  FitParamOn[MAXFITPARAMS];
int                      StartPos;
int                         EndPos;
float                 XScale;
Boolean               FitWholeFile = TRUE;
                      /*if FALSE, then user must select portion of file
                                                                 to be fit
via the mouse-LWP*/
int                     CurrentFunction;
main()
{
      int          myRsrc;

      InitGraf(&thePort);
      InitFonts();
      FlushEvents( everyEvent, 0 );
      InitWindows();
      InitMenus();
      TEInit();
      InitDialogs(0L);
      InitCursor();
      MaxApplZone();

      Stdio_MacInit(TRUE);         /*********!!!!!!!!!!!!! kill me
!!!*/

      OpenPlotWind();

/*
The following statement is included as a check to see if we can
access our program's resources.  When the project is run from
LightspeedC, the resource file <project name>.rsrc is automatically
opened.  When an application is built, these resources are
automatically merged with the application.
If you have followed the manual correctly, you haved named this
project "new project" and placed it on the same disk as the sources
for MiniEdit.  If this is so, the test is not necessary.

Please see the Addenda for more information on how LightspeedC
coordinates .rsrc files with projects and application builds.
*/

      if (GetResource('MENU', fileID)==0) {
            SysBeep(20);
            CantOpen();
            return;
      }

      SetUpFiles();
      SetUpCursors();
      SetUpMenus();
      SetUpWindows();
      if (! NOCOMMUNICATE) {                              /* added MJR*/
            InitSerialDrivers();
            ActuatorInit();
            BoxcarInit();
      }
      SetUpParamFiles();
      /*added MJR */
      while (MainEvent()) ;
```

```
}

int  MainEvent()
{
      EventRecord         myEvent;
      WindowPtr           whichWindow;
      Rect                r;

      MaintainCursor();
      MaintainMenus();
      SystemTask();
      TEIdle(TEH);
      if  (! NOCOMMUNICATE)  {
            /*read  and  echo  incoming  serial  data  stream  --  added  MJR
      */
            /*  Note  that  the  returned  string  is  unused
                  */
            GetReturnedStr(JunkStr255,  &actuatorPort,  TRUE,  DONTWAIT);
            /*EchoBoxcarOutput();
            read  and  echo  incoming  IEEE  data  stream  --  added  MJR  */
      }
      if  (GetNextEvent(everyEvent,  &myEvent))  {
            switch  (myEvent.what)  {
            case  mouseDown:
                  switch  (FindWindow(  myEvent.where,  &whichWindow  ))  {
                  case  inDesk:
                        SysBeep(10);
                        break;
                  case  inGoAway:
                        if  (ours(whichWindow))
                              if  (TrackGoAway(  CommWindPtr,
myEvent.where)  )
                                    DoFile(fmClose);
                        break;
                  case  inMenuBar:
                        return(  DoCommand(  MenuSelect(myEvent.where)  )
);
                  case  inSysWindow:
                        SystemClick(  &myEvent,  whichWindow  );
                        break;
                  case  inDrag:
                        if  (ours(whichWindow))
                        DragWindow(whichWindow,  myEvent.where,  &dragRect
);
                        break;
                  case  inGrow:
                        if  (ours(whichWindow))
                              MyGrowWindow(  whichWindow,  myEvent.where
);
                        break;
                  case  inContent:
                        if  (whichWindow  !=  FrontWindow())
                              SelectWindow(whichWindow);
                        else
                              if  (ours(whichWindow))
                                    DoContent(whichWindow,  &myEvent);
                        break;
                  default:  ;
```

```
                } /* end switch FindWindow */
                break;
            case  keyDown:
            case  autoKey:
                {
                register  char     theChar;

                theChar  =  myEvent.message  &  charCodeMask;
                if  ((myEvent.modifiers  &  cmdKey)  !=  0)
                    return(  DoCommand(  MenuKey(  theChar  )  ));
                else  {
                    if  (deviceSend  ==  DEVICESETACTUATOR)  {
                        SendActChar(theChar,&actuatorPort);
     /*added  MJR */
                    }  else  {
                        SendBoxChar(theChar,&boxcarPort);
     /*added  MJR */
                    }
                    TEKey(  theChar,  TEH  );
                    ShowSelect();
                    dirty  =  1;
                    }
                }
                break;
            case  activateEvt:
                if  (ours((WindowPtr)myEvent.message))  {
                    r=(*CommWindPtr).portRect;
                    r.top  =  r.bottom  -  (SBarWidth+1);
                    r.left  =  r.left  -  (SBarWidth+1);
                    InvalRect(&r);
                    if  (  myEvent.modifiers  &  activeFlag  )  {
                        TEActivate(  TEH  );
                        ShowControl(  vScroll  );
                        DisableItem(  myMenus[editM],  undoCommand
);

                        TEFromScrap();
                    }
                    else  {
                        TEDeactivate(TEH);
                        HideControl(  vScroll  );
                        ZeroScrap();
                        TEToScrap();
                    }
                }
                break;
            case  updateEvt:
                DealWithUpdates(myEvent);
                break;
            default:  ;
            }  /*  end  of  case  myEvent.what  */
     }  /*  if  */

     if  (RunInProgress  ==  INPROGRESS)  {               /*  added  MJR
     */
         DoDataScan();
         if  (ScanCounter  ==  NumOfScans)
             RunOver();
     }
```

```
	return(1);
}

SetUpMenus()
{
	int		i;

	myMenus[appleM] = NewMenu( appleID, "\p\024" );
	AddResMenu( myMenus[appleM], 'DRVR' );
	myMenus[fileM]  =  GetMenu(fileID);
	myMenus[editM]  =  GetMenu(editID);
	myMenus[communicateM]  =  GetMenu(communicateID); /* added MJR
	*/
	myMenus[datacollectionM]  =  GetMenu(datacollectionID);
	myMenus[fitM]  =  GetMenu(fitID);
	for ( (i=appleM); (i<NUMOFMENUS); i++ ) InsertMenu(myMenus[i], 0)
;

	DisableItem( myMenus[fileM], fmPrint );
		/* added MJR in eliminating printing	*/
	DisableItem( myMenus[fileM], fmPageSetUp );
		/* added MJR in eliminating printing	*/
	DisableItem( myMenus[datacollectionM], dcPause );
		/* Must START before you can PAUSE	*/
	DisableItem( myMenus[communicateM], cmSend );
		/* This option not yet installed		*/

	DrawMenuBar();
}

int DoCommand( mResult )
long mResult;
{
	int		theItem, temp;
	Str255		name;
	WindowPeek	wPtr;

	theItem = LoWord( mResult );
	switch (HiWord(mResult)) {
	case appleID:
		GetItem(myMenus[appleM], theItem, &name);
		OpenDeskAcc( &name );
		SetPort( CommWindPtr );
		break;
	case fileID:
		DoFile(theItem);
		break;
	case editID:
		if (SystemEdit(theItem-1)==0) {
			wPtr = (WindowPeek) FrontWindow();
			switch (theItem) {
			case cutCommand:
				TECut( TEH );
				dirty = 1;
				break;
			case copyCommand:
				TECopy( TEH );
```

```
                    break;
                case  pasteCommand:
                    TEPaste(  TEH  );
                    dirty  =  1;
                    break;
                case  clearCommand:
                    TEDelete(  TEH  );
                    dirty  =  1;
                    break;
                default:  ;
                }
                ShowSelect();
            }
            break;
        case  communicateID:
            DoCommunicationsMenu(theItem);
            break;
        case  datacollectionID:
            DoDataCollectionMenu(theItem);
            break;
        case  fitID:
            DoFitMenu(theItem);
            break;

    }
    HiliteMenu(0);
    return(1);
}

MaintainCursor()
{
    Point       pt;
    WindowPeek  wPtr;
    GrafPtr           savePort;

    if  (ours((WindowPtr)(wPtr=(WindowPeek)FrontWindow())))  {
        GetPort(  &savePort  );
        SetPort(  (GrafPtr)wPtr  );
        GetMouse(&pt);
        if  (  PtInRect(pt,  &(**TEH).viewRect  )    )
            SetCursor(  &editCursor);
        else  SetCursor(  &arrow  );
        SetPort(  savePort  );
    }
}

MaintainMenus()
{
    if  (  !(*(WindowPeek)CommWindPtr).visible  ||
            !ours(FrontWindow())  )  {
        EnableItem(  myMenus[fileM],  fmNew  );
        EnableItem(  myMenus[fileM],  fmOpen  );
        DisableItem(  myMenus[fileM],  fmClose  );
        DisableItem(  myMenus[fileM],  fmSave  );
        DisableItem(  myMenus[fileM],  fmSaveAs  );
        DisableItem(  myMenus[fileM],  fmRevert  );
        DisableItem(  myMenus[fileM],  fmPrint  );
        EnableItem(  myMenus[editM],  undoCommand  );
```

```
            EnableItem(  myMenus[editM],  cutCommand  );
            EnableItem(  myMenus[editM],  copyCommand  );
            EnableItem(  myMenus[editM],  clearCommand  );
      }
      else  {
            DisableItem(  myMenus[fileM],  fmNew  );
            DisableItem(  myMenus[fileM],  fmOpen  );
            EnableItem(  myMenus[fileM],  fmClose  );
            EnableItem(  myMenus[fileM],  fmSaveAs  );
      /*    EnableItem(  myMenus[fileM],  fmPrint  );     */
            if  (dirty)  {
                  EnableItem(  myMenus[fileM],  fmRevert  );
                  if(theFileName[0]!=0)  EnableItem(
myMenus[fileM],fmSave  );
                  else   DisableItem(  myMenus[fileM],  fmSave  );
            }
            else  {
                  DisableItem(  myMenus[fileM],  fmRevert  );
                  DisableItem(  myMenus[fileM],  fmSave  );
            }
            DisableItem(  myMenus[editM],  undoCommand  );
            if  ((**TEH).selStart==(**TEH).selEnd)  {
                  DisableItem(  myMenus[editM],  cutCommand  );
                  DisableItem(  myMenus[editM],  copyCommand  );
                  DisableItem(  myMenus[editM],  clearCommand  );
            }
            else  {
                  EnableItem(  myMenus[editM],  cutCommand  );
                  EnableItem(  myMenus[editM],  copyCommand  );
                  EnableItem(  myMenus[editM],  clearCommand  );
            }
      }
}

SetUpCursors()
{
      CursHandle  hCurs;

      hCurs  =  GetCursor(1);
      editCursor  =  **hCurs;
      hCurs  =  GetCursor(watchCursor);
      waitCursor  =  **hCurs;
}

ours(w)
WindowPtr  w;
{
      return(  (CommWindPtr!=NULL)  &&  (w==CommWindPtr)  );
}

CantOpen()
{
      Rect  r;

      SetRect(&r,  152,  60,  356,  132);
      SetPort((CommWindPtr  =  NewWindow(0L,&r,"\p",1,1,-1L,0,0L)));
      TextFont(0);
      MoveTo(  4,  20  );
```

```
	DrawString("\pCan't  open  resource  file.");
	MoveTo( 4, 40 );
	DrawString("\pClick  mouse  to  exit.");
	do  {
	} while  (  !Button()  );
}

DealWithUpdates(Event)
EventRecord Event;
{
	WindowPtr	UpDateWindow,  TempPort;
	Rect		TempRect;
	Str255			tempstr;
	int			i;

/*if  (ours((WindowPtr)Event.message))  UpdateWindow(CommWindPtr);  */


	GrafPtr		savePort;

	UpDateWindow  =  (WindowPtr)  Event.message;
	GetPort(&TempPort);				/*Save  the  current  port*/

	SetPort(UpDateWindow);		/*set  the  port  to  one  in  Evt.msg*/
	BeginUpdate(UpDateWindow);

	switch  ((*((WindowPeek)UpDateWindow)).windowKind)  {
		case  COMMWINDKIND:
			UpdateWindow(UpDateWindow);
			break;
		case  EDITWINDKIND:
			break;
		case  PLOTWINDKIND:
			/*DebugStr("\pAbout  to  do  an  update");
			DrawPlotWind(UpDateWindow);*/
			break;
	}

	EndUpdate(UpDateWindow);
	SetPort(TempPort);				/*restore  to  the  previous
port*/
}
```

```
/*
	File Name:		Decode.c
	Version:		1.0
	Project Name:	DataAcquisition
	Authors:		Mark J. Rosker
	Compiler:		Lightspeed C 3.01

	Utility:		This subroutine takes the TEHandle TEH and
					decodes it into an X and Y array, which are
					stuffed into the
					structure DP.  The number of data points is
					returned as NumOfDataPts.

	This is done as follows:
	(1)  Each '\n' character signals the end of a line.  Function
	ReadALine gets each line.(this is like a sequential formatted
	FORTRAN read)
	(2) The first two lines are ignored.  This is the header stuff.
	(3) Data arrays are taken to be the first two values.  Presently,
	scanf is used to decode, and it seems to do a good job.  Done in
	function GetTwoValues.

	Notes:			Note that the TEH is just used as a buffer to
					store the input text. It is not changed or
					updated in any way.
*/

#include "MacTypes.h"
#include "TextEdit.h"
#include "Data Acquisition.h"


DecodeTEHToData(TEH)
TEHandle	TEH;
{
	char				**ACharHdl;
	char				*FirstCharPtr;
	Str255					AStr255;
	long				Counter;
	float				x,y;
	int						DPCounter;
	extern		float		*XData;
	extern		float		**YData;
	extern		int				NumYRows, CurrYRow;
	extern		int				NumOfDataPts;

	ACharHdl = TEGetText(TEH);
	FirstCharPtr = *ACharHdl;
	/* remember to what you pointed at at start	*/

	/* First time thru: Find out how many lines there are */
	Counter = 0L;
	DPCounter = 0;

	ReadALine(TEH,ACharHdl,&Counter,AStr255);
	/* read the '*'  -- and then ignore it   */
	ReadALine(TEH,ACharHdl,&Counter,AStr255);
	/* read the axes titles -- and then ignore it   */
	while (Counter < (**TEH).teLength) {
		ReadALine(TEH,ACharHdl,&Counter,AStr255);
		DPCounter++;
	}
	NumOfDataPts = DPCounter;

	/* Second time thru: Actually do the read */
	*ACharHdl = FirstCharPtr;/* point back to the original character*/
	Counter = 0L;

	ReadALine(TEH,ACharHdl,&Counter,AStr255);
	/* read the '*'  -- and then ignore it   */
	ReadALine(TEH,ACharHdl,&Counter,AStr255);
	/* read the axes titles -- and then ignore it   */

	DPCounter = 0;
	NumYRows = 1;
	CurrYRow = 0;

	MakeDataArrays();
	/*Only make the arrays after NumOfDataPts is known    */

	while (Counter < (**TEH).teLength) {
		ReadALine(TEH,ACharHdl,&Counter,AStr255);
		GetTwoValues(AStr255,&x,&y);
		XData[DPCounter]		= x;
		YData[CurrYRow][DPCounter]	= y;
		DPCounter++;
	}
	NumOfDataPts = DPCounter;

	*ACharHdl = FirstCharPtr;
	/* point back to the original character		*/
}

ReadALine(TEH, ACharHdl, CounterPtr, ALine)	/* returns a C string */
TEHandle	TEH;
char		**ACharHdl;
long		*CounterPtr;
Str255			ALine;
{
	int	j;
	for (j = 0; *CounterPtr < (**TEH).teLength || j >= 254;
	(*CounterPtr)++, (*ACharHdl)++)  {
		if (**ACharHdl == '\n' || **ACharHdl == '\r') {
			(*CounterPtr)++;	/* skip this character  */
			(*ACharHdl)++;
			break;
		}
		ALine[j++] = **ACharHdl;
	}
	ALine[j] = '\0';
}

GetTwoValues(Line,Value1,Value2)/* assumes Line is a C String    */
Str255		Line;
float *Value1,*Value2;
{
        sscanf(Line,"%g%g",Value1,Value2);
}
```

```
/*
      File Name:          dialogRoutines.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Mark J. Rosker
      Compiler:           Lightspeed C 3.01

      Utility:            This section contains the routines that handle
                          the dialogs used in the graphic interface
*/

#include  "ControlMgr.h"
#include  "DialogMgr.h"
#include  "EventMgr.h"
#include  "MacTypes.h"
#include  "Quickdraw.h"
#include  "ToolboxUtil.h"

extern  Str255     JunkStr255;
extern  Handle     JunkHandle;
extern  int        JunkInt;
extern  Rect       JunkRect;

HideItem(theDialog,  ItemNo)
DialogPtr   theDialog;
int                ItemNo;
{
      GetDItem(theDialog,  ItemNo,  &JunkInt,  &JunkHandle,  &JunkRect);
      InsetRect(&JunkRect,  -4,  -4);
      EraseRect(&JunkRect);
      InsetRect(&JunkRect,  4,  4);
      OffsetRect(&JunkRect,  -500,  0);
      SetDItem(theDialog,  ItemNo,  JunkInt,  JunkHandle,  &JunkRect);
}

ShowItem(theDialog,  ItemNo)
DialogPtr   theDialog;
int                ItemNo;
{
      GetDItem(theDialog,  ItemNo,  &JunkInt,  &JunkHandle,  &JunkRect);
      OffsetRect(&JunkRect,  500,  0);
      SetDItem(theDialog,  ItemNo,  JunkInt,  JunkHandle,  &JunkRect);
}

HiliteItem(theDialog,  ItemNo,  HiliteMode)
DialogPtr   theDialog;
int                ItemNo,HiliteMode;
{
      GetDItem(theDialog,  ItemNo,  &JunkInt,  &JunkHandle,  &JunkRect);
      HiliteControl(JunkHandle,  HiliteMode);
}

frameOkbox(theDialog)
DialogPtr   theDialog;
{
      GetDItem(theDialog,  1,   &JunkInt,  &JunkHandle,  &JunkRect);
      InsetRect(&JunkRect,  -4,  -4);
      PenSize(3,  3);
```

```
        FrameRoundRect(&JunkRect,  16,  16);
        PenSize(1,  1);
}

ItemCheckMark(theDialog,  ItemNo,  ChkMark)
DialogPtr   theDialog;
int                 ItemNo,  ChkMark;
{
        GetDItem(theDialog,  ItemNo,  &JunkInt,  &JunkHandle,  &JunkRect);
        SetCtlValue(JunkHandle,  ChkMark);
}

ShowIcon(theDialog,  ItemNo,  IconID)
/*
        Given a dialog, this routine puts a new icon of resource
        ID IconID at item ItemNo.
 */

DialogPtr   theDialog;
int                 ItemNo,  IconID;
{
        GetDItem(theDialog,  ItemNo,  &JunkInt,  &JunkHandle,  &JunkRect);
        JunkHandle  =  GetIcon(IconID);
        SetDItem(theDialog,  ItemNo,  JunkInt,  JunkHandle,  &JunkRect);
}


SetUpTEDlogInt(theDialog,itemHit,Value)
DialogPtr   theDialog;
int                 itemHit;
int         Value;
/*
        Set-up dialog box TextEdit stuff for item number itemHit of
        theDialog.
        The current value of the integer is Value.
*/

{
        Str255              theText;
        Handle              theTextHdl;
        int                 theType;
        Rect        txtBox;

        GetDItem(theDialog,  itemHit,  &theType,  &theTextHdl,  &txtBox);
        sprintf(theText,"%d\0",Value);
        CtoPstr(theText);
        SetIText(theTextHdl,theText);
        SelIText(theDialog,itemHit,0,32767);

}
SetUpTEDlogLInt(theDialog,itemHit,Value)
DialogPtr   theDialog;
int                 itemHit;
long        Value;

/*
         Set-up dialog box TextEdit stuff for item number itemHit of
         theDialog.
        The current value of the integer is Value.
*/

{
      Str255             theText;
      Handle             theTextHdl;
      int                theType;
      Rect         txtBox;

      GetDItem(theDialog, itemHit, &theType, &theTextHdl, &txtBox);
      sprintf(theText,"%ld\0",Value);
      CtoPstr(theText);
      SetIText(theTextHdl,theText);
      SelIText(theDialog,itemHit,0,32767);

}
SetUpTEDlogDouble(theDialog,itemHit,Value)
DialogPtr   theDialog;
int                itemHit;
double             Value;

/*
      Set-up dialog box TextEdit stuff for item number itemHit of
      theDialog.
      The current value of the floating pt. number is Value.
*/

{
      Str255             theText;
      Handle             theTextHdl;
      int                theType;
      Rect         txtBox;

      GetDItem(theDialog, itemHit, &theType, &theTextHdl, &txtBox);
      sprintf(theText,"%g\0",Value);
      CtoPstr(theText);
      SetIText(theTextHdl,theText);
      SelIText(theDialog,itemHit,0,32767);

}

DoTEDlogInt(theDialog,itemHit,ValuePtr)
DialogPtr   theDialog;
int                itemHit;
int         *ValuePtr;
/*
      Do the TextEdit stuff for item number itemHit of theDialog.
      Returns the current value of the integer pointed by ValuePtr.
 */

{
      Str255             theText;
      Handle             theTextHdl;
      int                theType;
      Rect         txtBox;

      int                atoi();
	GetDItem(theDialog, itemHit, &theType, &theTextHdl, &txtBox);
	GetIText(theTextHdl, theText);
	/* tells us the text the user entered */
	if (IsAnInt(theText)) {
		PtoCstr(theText);
		*ValuePtr = atoi(theText);
		CtoPstr(theText);
	} else {
		SysBeep(2);
		sprintf(theText,"%d\0",*ValuePtr);
		CtoPstr(theText);
		SetIText(theTextHdl, theText);
	}
}
DoTEDlogLInt(theDialog,itemHit,ValuePtr)
DialogPtr	theDialog;
int			itemHit;
long		*ValuePtr;
/*
	Do the TextEdit stuff for item number itemHit of theDialog.
	Returns the current value of the integer pointed by ValuePtr.
 */

{
	Str255			theText;
	Handle			theTextHdl;
	int			theType;
	Rect		txtBox;

	long			atol();

	GetDItem(theDialog, itemHit, &theType, &theTextHdl, &txtBox);
	GetIText(theTextHdl, theText);
	/* tells us the text the user entered */
	if (IsAnInt(theText)) {
		PtoCstr(theText);
		*ValuePtr = atol(theText);
		CtoPstr(theText);
	} else {
		SysBeep(2);
		sprintf(theText,"%ld\0",*ValuePtr);
		CtoPstr(theText);
		SetIText(theTextHdl, theText);
	}
}
DoTEDlogDouble(theDialog,itemHit,ValuePtr)
DialogPtr	theDialog;
int			itemHit;
double			*ValuePtr;
/*
 * Do the TextEdit stuff for item number itemHit of theDialog.
 * Returns the current value of the integer pointed by ValuePtr.
 */

{
	Str255			theText;
	Handle			theTextHdl;
	int			theType;
	Rect		txtBox;
	float		silly;		/* need to use a float  */

	GetDItem(theDialog, itemHit, &theType, &theTextHdl, &txtBox);
	GetIText(theTextHdl, theText);
	/* tells us the text the user entered */
	PtoCstr(theText);
	sscanf(theText,"%g",&silly);
	*ValuePtr = silly;
	CtoPstr(theText);
}

ToggleItemCheckMark(theDialog,itemHit,LogicalStatePtr)
DialogPtr	theDialog;
int		itemHit;
Boolean		*LogicalStatePtr;
{
	if (*LogicalStatePtr) {
		ItemCheckMark(theDialog, itemHit, 0);
		*LogicalStatePtr = FALSE;
	} else {
		ItemCheckMark(theDialog, itemHit, 1);
		*LogicalStatePtr = TRUE;
	}
}
```

```
/*
      File Name:          file.mini.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            From the MiniEdit project in LightSpeedC
      Compiler:           Lightspeed C 3.01

      Utility:            This section manages the routines from File and
                          Edit menus

      Notes:              Addapted by Mark J. Rosker
*/

#include "Data Acquisition.h"

Str255              theFileName;
static int theVRefNum;

extern TEHandle   TEH;
extern WindowPtr  CommWindPtr;
extern char         dirty;
extern int        NumOfDataPts;

SetUpFiles()
{
      pStrCopy("\p", theFileName);
      theVRefNum = 0;
}

int DoFile( item )
int         item;

{
      int   vRef, refNum, io;
      Str255      fn;
      extern      Boolean                 DataTakenFlag;/* added mjr*/
      extern      Boolean           FirstFit;
      extern      float       *XData;
      extern      float       **YData;
      extern      int               NumYRows, CurrYRow;
      extern      int               NumOfDataPts;

      switch (item) {
      case fmNew:
            SetWTitle( CommWindPtr, "\pUntitled");
            ShowWindow( CommWindPtr );
            dirty = 0;
            break;
      case fmOpen:
            if (OldFile( &fn, &vRef ))
                  if (FSOpen( &fn, vRef, &refNum)==noErr) {
                        if (ReadFile( refNum, TEH )) {
                              pStrCopy(fn, theFileName);
                              theVRefNum = vRef;
                              SetWTitle(CommWindPtr, theFileName);
                              dirty = 0;
                              FirstFit = TRUE;
                        }
                        if  (FSClose(refNum)==noErr)  ;
                        ShowWindow(  CommWindPtr  );
                        DecodeTEHToData(TEH);   /* added mjr
        */
                        DataTakenFlag  =  TRUE;
                        DrawPlotWind(TRUE,XData,YData,NumOfDataPts,
                              NumYRows,CurrYRow,NIL); /*added  mjr*/
                  }
                  else FileError( "\pError opening ", fn );
            break;
        case  fmClose:
            if (dirty) {
                  switch (Advise("\pDo you want to save the changes to

            your file?")) {
                  case  aaSave:
                        if  (theFileName[0]==0)  {
                              fn[0]  =  0;
                              if  (!SaveAs(&fn,  &vRef))  return(0);
                        }
                        else  if  (!SaveFile(  theFileName,  theVRefNum  ))
                              return(0);
                        break;
                  case  aaCancel:  return(0);
                  case  aaDiscard:  dirty  =  0;
                  }
            }
            CloseCommWindPtr();
            break;
        case  fmSave:
            if  (theFileName[0]==0)  goto  saveas;
            SaveFile(theFileName,  theVRefNum);
            break;
        case  fmSaveAs:
saveas:
            fn[0]  =  0;
            if (SaveAs( &fn, &vRef )) {
                  pStrCopy(fn,  theFileName);
                  theVRefNum  =  vRef;
                  SetWTitle(CommWindPtr,  theFileName);
            }
            break;
        case  fmRevert:
            HidePen();
            TESetSelect(  0,  (**TEH).teLength,  TEH  );
            ShowPen();
            TEDelete(TEH);
            if  ((theFileName[0]!=0)  &&
                  (FSOpen(  &theFileName,  theVRefNum,  &refNum)==noErr))  {
                  dirty  =  !ReadFile(  refNum,  TEH  );
                  /*  I  don't  check  for  bad  read!  */
                  if  (FSClose(refNum)==noErr)  ;
            }
            ShowWindow(  CommWindPtr  );
            UpdateWindow(  CommWindPtr  );
            break;

/*  Note:  PageSetUp  and  Print  were  de-allowed  by  MJR
```

```
                                (so that printer port could be used)
     */
     case   fmPageSetUp:
            /*DoPageSetUp();*/
            break;
     case   fmPrint:
     /*     PrintText((**TEH).hText,(long)(**TEH).teLength,
                 (GrafPtr)CommWindPtr,StringWidth("\pmmmm"),  0L);*/
            break;
     case   fmQuit:
            if  (DoFile(fmClose))  {
                 CloseSerialDrivers();
                 ExitToShell();
            }
     }
     return(1);
}

static Point SFGwhere = { 90, 82 };
static Point SFPwhere = { 106, 104 };
static SFReply reply;

SaveAs( fn, vRef )
Str255      *fn;
int         *vRef;
{
     int  refNum;

     if  (NewFile(fn,  vRef))
          if  (CreateFile(fn,  vRef,  &refNum))  {
                WriteFile(refNum,  (*(**TEH).hText),

     (long)(**TEH).teLength);
                FSClose(  refNum  );
                dirty = 0;
                return(1);
          }
          else  {
                FileError("\pError  creating  file  ",  fn);
          }
     return(0);
}

SaveFile(  fn,  vrn  )
Str255      fn;
int         vrn;
{
     int  refNum;

     if  (FSOpen(  fn,  vrn,  &refNum  )==noErr)  {
          WriteFile(refNum,  (*(**TEH).hText),  (long)(**TEH).teLength);
          dirty = 0;
          FSClose(  refNum  );
          return(1);
     }
     else  FileError("\pError  opening  file  ",  fn);
     return(0);
}
```

```
NewFile( fn, vRef )
Str255      *fn;
int         *vRef;
{
      SFPutFile(SFPwhere, "\p", fn, 0L, &reply);
      if (reply.good) {
            pStrCopy(reply.fName, fn);
            *vRef = reply.vRefNum;
            return(1);
      }
      else return(0);
}

OldFile( fn, vRef )
Str255      *fn;
int         *vRef;
{
      SFTypeList myTypes;

      myTypes[0]='CGTX';
      myTypes[1]='TEXT';
      SFGetFile( SFGwhere, "\p", 0L, 2, myTypes, 0L, &reply );
      if (reply.good) {
            pStrCopy( reply.fName, fn );
            *vRef = reply.vRefNum;
            return(1);
      }
      else return(0);
}

CreateFile( fn, vRef, theRef )
Str255      *fn;
int         *vRef;
int         *theRef;
{
      int io;

      io=Create(fn, *vRef, '????', 'CGTX');
      if((io==noErr)||(io==dupFNErr)) io = FSOpen( fn, *vRef,theRef );
      return( (io==noErr) || (io=dupFNErr) );
}

WriteFile( refNum, p, num )
int         refNum;
char  *p;
long  num;
{
      int io;                 /*    a real application would want to
check
                                    this return code for failures */

      io=FSWrite( refNum, &num, p );
}

ReadFile( refNum, textH )
int         refNum;
TEHandle textH;
```

```
{
	char  buffer[256];
	long  count;
	int		io;

	TESetSelect(0, (**textH).teLength, textH);
	TEDelete( textH );
	do {
		count = 256;
		io = FSRead( refNum, &count, &buffer );
		TEInsert( &buffer, count, textH );
	} while (io==noErr);
	return( io==eofErr );
}

pStrCopy( p1, p2 )
register char *p1, *p2;
/* copies a pascal string from p1 to p2 */
{
	register int len;

	len = *p2++ = *p1++;
	while (--len>=0) *p2++=*p1++;
}

FileError( s, f )/* assumes both s and f are now Pascal strings   */
Str255 s, f;
{
	ParamText(s, f,"\p", "\p");
	NoteAlert( ErrorAlert, 0L );
}

Advise( s )		/* assumes s is now Pascal string   */
Str255 s;
{
	ParamText(s,"\p","\p","\p");
	return( NoteAlert(AdviseAlert, 0L) );
}
```

```
/*
      File Name:          Fit.Dlog.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Mark J. Rosker and Lawrence W. Peng
      Compiler:           Lightspeed C 3.01

      Utility:            This file takes care of the graphics interface
                          of the NLS fitting routine of the project
*/

#include  "MacTypes.h"
#include  "DialogMgr.h"
#include  "Data Acquisition.h"
#include  "least_squares.h"

extern      Str255          JunkStr255;
extern      WindowPtr   PlotWindPtr;
extern   Boolean   FitWholeFile;

/*
Has been adjusted to correctly display the a[] parameters as unit
offset. LWP 10/21/88
*/

int   DoFitDlog(a,FitParamOn,NumOfParams)
double      a[];
Boolean     FitParamOn[];
int         NumOfParams;
{
      DialogPtr   theDialog;
      int               itemHit;
      WindowPtr   tmpWindPtr;
      extern            DialogRecord      DlogRec;
      Str255            theText;
      Handle            theTextHdl;
      Int               theType;
      Rect        txtBox;
      register     int      i;
      Boolean       tempFitWholeFile;
      double            tempA[MAXFITPARAMS];
      Boolean           tempFitParamOn[MAXFITPARAMS];

      tempFitWholeFile  =  FitWholeFile;

      GetPort(&tmpWindPtr);         /* Point at current window    */

      theDialog  =  GetNewDialog(FITDLOGID, &DlogRec, (WindowPtr)(-1) );
      SetPort(theDialog);

      a[0]=0.0;
      for (i = 0; i< 7; i++) {
            if (i < NumOfParams) {
                  tempFitParamOn[i]  =  FitParamOn[i];
                  tempA[i+1]  =  a[i+1];
                  SetUpTEDlogDouble(theDialog,10  +  i,tempA[i+1]);
            if  (tempFitParamOn[i])  {
                  ItemCheckMark(theDialog, 3 + i, 1);
```

```
                }else
                        ItemCheckMark(theDialog, 3 + i, 0);
            } else {
                        HideDItem(theDialog, 3 + i);
                        HideDItem(theDialog, 10 + i);

/*
047 has the above two statements as 'HideItem'. However, the display
did not work correctly if the desk accessory PYRO ever made the
screen dark. Therefore, direct MacIntosh calls have been used
instead.  LWP 10/22/88
*/

            }
        }

        ItemCheckMark(theDialog,23,tempFitWholeFile);
        /*Default to fitting entire data file-LWP*/

        ShowWindow(theDialog);
        frameOkbox(theDialog);

        do{
            ModalDialog(NIL, &itemHit);
            /*calls filter before handling*/
            switch (itemHit){
                case 1 :                        /* Ok button
        */
                case 2 :                        /* Cancel button  */
                        break;
                case 3:     /* Parameter check boxes      */
                case 4:
                case 5:
                case 6:
                case 7:
                case 8:
                case 9:
                        if (tempFitParamOn[itemHit - 3]) {
                            ItemCheckMark(theDialog, itemHit, 0);
                            tempFitParamOn[itemHit - 3] = FALSE;
                        } else {
                            ItemCheckMark(theDialog, itemHit, 1);
                            tempFitParamOn[itemHit - 3] = TRUE;
                        }
                        break;
                case 10:                            /* parameter*/
                case 11:
                case 12:
                case 13:
                case 14:
                case 15:
                case 16:
                        DoTEDlogDouble(theDialog,itemHit,&tempA[itemHit-
9]);

                        break;
                case 23:
                        if(tempFitWholeFile) {
                            ItemCheckMark(theDialog,23,0);
```

```
                                tempFitWholeFile  =  FALSE;
                                }else{
                                ItemCheckMark(theDialog,23,1);
                                tempFitWholeFile  =  TRUE;
                                }
                     default:
                          break;
                }                                  /*the case of what item was
hit*/
      }while (itemHit != 1 && itemHit != 2);

      CloseDialog(theDialog);
      SetPort(tmpWindPtr);

      if (itemHit != 2) {                          /* Unless it was cancel...
      */
           for (i = 0; i < NumOfParams; i++) {
                FitParamOn[i]  =  tempFitParamOn[i];
                a[i+1]         =  tempA[i+1];
                }
           FitWholeFile  =  tempFitWholeFile;

           return(TRUE);
      } else {
           return(FALSE);
      }
}
```

```
/*
        File Name:          funcs.c
        Version:            1.0
        Project Name:       DataAcquisition
        Authors:            Marcos Dantus, Mark J. Rosker,
                                 Lawrence W. Peng
        Compiler:           Lightspeed C 3.01

        Utility:            This file contains the functional forms to
                            be used in the NLS fitting routine

        Notes:              All functions in this file should have identical
                            format so that they may be used interchangably.
                            Try to use the same naming convention for new
                            funcs, too.
                            THE PARAMETER LIST STARTS WITH UNIT OFFSET,
                            NOT ZERO OFFSET!!!
*/


#define              EVALUE        7000.0
/* According to the theory for AZ's Model         */
#define              SQRTPI        1.772453851      /* sqrt of pi      */
#include  "math.h"
#include  "MacTypes.h"
#define       MAXFITPARAMS              20
/* Set to largest number of fit parameters        */

extern  int  CountEm;

void    LineFunc(x,a,yPtr,dyda,na)          /*Fit to a line*/
double   x,*yPtr;
double   a[],dyda[];
int       na;
{
        if (na != 2)
             DebugStr("\pTrouble in LineFunc.");

        *yPtr = x*a[1] + a[2];

        dyda[1] = x;
        dyda[2] = 1.0;
}

void    SingExpFunc(x,a,yPtr,dyda,na)  /*Single exponential function*/
double   x,*yPtr;
double   a[],dyda[];
int       na;
{
        double   arg;

        if (na != 4)
             DebugStr("\pTrouble in SingleExpFunc.");

        arg = ((x-a[2])/a[3]);

        if(x < a[2]) {*yPtr = a[4];}
             else{
```

```
        *yPtr  =  a[1]*(exp(-arg))  +  a[4];}

        if(x  <  a[2]){
        dyda[1]  =dyda[2]  =dyda[3]  =dyda[4]  =  0.0;
        }else{

        dyda[1]  =  exp(-arg);
        dyda[2]  =  a[1]*exp(-arg)/a[3];
        dyda[3]  =  a[1]*arg*exp(-arg)/a[3];
        dyda[4]  =  1.0;}
}

void    BiExpFunc(x,a,yPtr,dyda,na)

double   x,*yPtr;
double   a[],  dyda[];
int      na;
{
        double  arg1,arg2,fac;

        if  (na  != 6)
                DebugStr("\pTrouble  in  DoubleExpFunc.");

        arg1  =  (x-a[4])/a[2];
        arg2  =  (x-a[4])/a[3];

        if(x  <  a[4])  *yPtr  =  a[6];  else
        *yPtr  =  a[1]*(exp(-arg1)  +  a[5]*exp(-arg2))  +  a[6];
        fac  =  (exp(-arg1)  +  a[5]*exp(-arg2));

        if(x  <  a[4]){
        dyda[1]  =dyda[2]  =dyda[3]  =dyda[4]  =dyda[5]  =dyda[6]  =  0.0;
        }else
        dyda[1]  =  fac;
        dyda[2]  =  a[1]*arg1*exp(-arg1)/a[2];
        dyda[3]  =  a[1]*a[5]*arg2*exp(-arg2)/a[3];
        dyda[4]  =  a[1]*(exp(-arg1)/a[2]  +a[5]*exp(-arg2)/a[3]);
        dyda[5]  =  a[1]*exp(-arg2);
        dyda[6]  =  1.0;
}

void  SinusoidFunc(x,a,yPtr,dyda,na)
/*  NOT  BEING  USED  */
double       x,*yPtr;
double       a[],  dyda[];
int          na;
{
        double       arg,  ex,  fac;

}

void  GaussianFunc(x,a,yPtr,dyda,na)
/*  gaussian  plus  a  constant.   na  had  better  be  4       */
double       x,*yPtr;
double       a[],  dyda[];
int          na;
{
        double       arg,  ex,  fac;
        if (na != 4)
                DebugStr("\pTrouble  in  GaussianFunc.");

        arg = (x - a[2]) / a[3];
        ex    =  exp(-arg*arg);
        *yPtr = a[1] * ex + a[4];

        fac   = a[1] * ex * 2.0 * arg;

        dyda[1]  =  ex;
        dyda[2]  =  fac / a[3];
        dyda[3]  =  fac * arg / a[3];
        dyda[4]  =  1.0;
}

void  LorentzianFunc(x,a,yPtr,dyda,na)
double      x,*yPtr;
double      a[], dyda[];
int         na;
{
        double      arg, fac, denom;

        if (na != 4)
                DebugStr("\pTrouble  in  LorentzianFunc.");

        arg   = (x - a[2]) / a[3];
        denom = 1.0 + (arg*arg);
        *yPtr        = a[1] / denom + a[4];

        fac   = a[1]  * 2.0 * arg / denom;

        dyda[1]  =  1.0 / denom;
        dyda[2]  =  fac / a[3];
        dyda[3]  =  fac * arg / a[3];
        dyda[4]  =  1.0;
}

void  HypSecantFunc(x,a,yPtr,dyda,na)
double      x,*yPtr;
double      a[], dyda[];
int         na;
{
        double      arg, fac, sech, My_tanh, sech2;

        if (na != 4)
                DebugStr("\pTrouble  in  HypSecantFunc.");

        arg   = (x - a[2]) / a[3];
        sech  = 2.0 / (exp(arg) + exp(-arg));
        My_tanh     = (exp(arg) - exp(-arg)) / (exp(arg) + exp(-arg));
        sech2 = sech * sech;
        *yPtr        = a[1] * sech2 + a[4];

        fac   = a[1] * sech2 * 2.0 * My_tanh;

        dyda[1]  =  sech2;
        dyda[2]  =  fac / a[3];
```

```
	dyda[3] = fac * arg / a[3];
	dyda[4] = 1.0;
}

void  ExpStepFunc(x,a,yPtr,dyda,na)
double		x,*yPtr;
double		a[], dyda[];
int		na;
{
	double		arg, ex, fac;

	CountEm++;

	if (na != 4)
		DebugStr("\pTrouble in ExpStepFunc.");

	arg	= (x - a[2]) / a[3];
	if (arg < 0)
		ex		= 1.0 - exp(arg);
	else
		ex		= 0.0;
	*yPtr		= a[1] * ex + a[4];

	fac	= a[1] * exp(arg);

	if (arg < 0) {
		dyda[1] = ex;
		dyda[2] = fac / a[3];
		dyda[3] = fac * arg / a[3];
	} else {
		dyda[1] = 0.0;
		dyda[2] = 0.0;
		dyda[3] = 0.0;
	}
	dyda[4] = 1.0;
}

void  AZmodelFunc(x,a,yPtr,dyda,na)
double		x,*yPtr;
double		a[], dyda[];
int		na;
{
	double		arg, ex, fac, denom, sech, My_tanh, sech2;
	double		amp, offset, alpha, bline, beta, gamma, amplitude,
dummy;

	if (na != 6)
		DebugStr("\pTrouble in AZmodelFunc.");

	/*	y(x;a) is the equation number (12) of the Bersohn and
	Zewail (1987) paper */
	/*	Definition of the variables	*/
	a[3] = fabs(a[3]);		/*	beta should always be positive */
	amp			= a[6];
	offset			= a[1];
	alpha		= a[2];
	bline		= a[5];
	beta		= fabs(a[3]);
```

```
        gamma       = a[4];
        arg   = (x - offset) * alpha;
        if (a[6] >= 0.0)
             arg = - arg;
        if (arg >= 0.0)
             sech = 1.0;
        else
             sech  = 2.0 / (exp(arg) + exp(-arg));

        My_tanh     = (exp(arg) - exp(-arg)) / (exp(arg) + exp(-arg));
        sech2 = sech * sech;
        fac         = EVALUE * EVALUE * (sech2 - beta) * (sech2 - beta);
        amplitude = amp * gamma * gamma;
        denom = ((gamma * gamma) + fac);
        (*yPtr)  =  amplitude/denom;
        dummy = 2.0 * (*yPtr) * fac / denom;

        dyda[6]     =  (gamma * gamma) / denom;
        dyda[1]     = - 2.0 * dummy * sech2 * My_tanh * alpha
             / (sech2 - beta);
        dyda[2]     = 2.0 * dummy * sech2 * My_tanh * (x - offset)
             / (sech2 - beta);
        dyda[5]     = 1.0;
        dyda[3]     = 2.0 * (*yPtr) * fac/(denom * (sech2 - beta));
        dyda[4]     = 2.0 * (*yPtr) * ((1.0 / gamma) - (gamma / denom));

        (*yPtr) = (*yPtr) + bline;

        }

void  MJRmodelFunc(x,a,yPtr,dyda,na)
double      x,*yPtr;
double      a[], dyda[];
int         na;
{
        double      arg;
        double        alpha, beta;

        double      erfc();
        double      s, erfcb, erfcab, expterm;

                         /* the derivations are in my notes  */
        if (na != 5)
             DebugStr("\pTrouble  in  MJRmodelFunc.");

        arg   = -x + a[2];
        alpha = a[5] / (2.0 * a[3]);
        beta  = - arg / a[5];

        erfcb =     erfc(beta);
        erfcab      =     erfc(alpha + beta);
        expterm     =     exp(alpha*(alpha + 2.0*beta));
        s           =     SQRTPI * (erfcb - expterm*erfcab) / 2.0;

        *yPtr       = a[1] * s + a[4];

        dyda[1] = s;
        dyda[2] = a[1] * SQRTPI * alpha * expterm * erfcab / a[5];
```

```
	dyda[3] = -a[1] * alpha * alpha * SQRTPI * expterm * erfcab
		/ a[3];
	dyda[4] = 1.0;
	dyda[5] = a[1]*alpha*(exp(-beta*beta) - alpha*SQRTPI
		*expterm*erfcab) / a[5];
}

void tempFunc(x,a,yPtr,dyda,na)	/* temp.; like bz function*/
double		x,*yPtr;
double		a[], dyda[];
int		na;
{
	double		arg, ex, fac, denom, sech, My_tanh, sech2;
	double		amp, offset, alpha, bline, beta, gamma, amplitude,
dummy;

	if (na != 5)
		DebugStr("\pTrouble in tempFunc.");

	/*	y(x;a) is the equation number (12) of the Bersohn and
	Zewail (1987) paper */
	/*	Definition of the variables	*/
	/*a[3] = fabs(a[3]);		beta should always be positive */
	amp			= a[5];
	offset			= a[1];
	alpha		= a[2];
	bline		= a[4];
	beta		= 0.0;
	gamma		= a[3];
	arg	= (x - offset) * alpha;
	if (a[6] >= 0.0)
		arg = - arg;
	if (arg >= 0.0)
		sech = 1.0;
	else
		sech = 2.0 / (exp(arg) + exp(-arg));

	My_tanh		= (exp(arg) - exp(-arg)) / (exp(arg) + exp(-arg));
	sech2 = sech * sech;
	fac		= EVALUE * EVALUE * (sech2 - beta) * (sech2 - beta);
	amplitude = amp * gamma * gamma;
	denom = ((gamma * gamma) + fac);
	(*yPtr) = amplitude/denom;
	dummy = 2.0 * (*yPtr) * fac / denom;

	if (arg < 0.0) {
		dyda[5]		=  (gamma * gamma) / denom;
		dyda[1]		= - 2.0 * dummy * sech2 * My_tanh * alpha
			/ (sech2 - beta);
		dyda[2]		= 2.0 * dummy * sech2 * My_tanh * (x - offset)
			/ (sech2 - beta);
		dyda[4]		= 1.0;

	/*	dyda[3]		= 2.0 * (*yPtr) * fac/(denom * (sech2 -
beta));*/
		dyda[3]		= 2.0 * (*yPtr) * ((1.0 / gamma) - (gamma /
denom));
	} else {
```

```
            dyda[5]     = (gamma * gamma) / denom;
            dyda[1]     = 0.0;
            dyda[2]     = 0.0;
            dyda[4]     = 1.0;

            dyda[3]     = 2.0 * (*yPtr) * ((1.0 / gamma) - (gamma /
denom));
        }
        (*yPtr) = (*yPtr) + bline;
}
```

```
/*
      File Name:          gaussj.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            subroutine from Numerical Recipes
      Compiler:           Lightspeed C 3.01

      Utility:            This file uses the Gauss Jordan method for
                          elimination (matrix math)

      Note:               In this modified version, double is used
                          instead of float
*/

#include <math.h>

#define SWAP(a,b) {double temp=(a);(a)=(b);(b)=temp;}

void gaussj(a,n,b,m)
double **a,**b;
int n,m;
{
      int *indxc,*indxr,*ipiv;
      int i,icol,irow,j,k,l,ll,*ivector();
      double big,dum,pivinv;
      void nrerror(),free_ivector();

      indxc=ivector(1,n);
      indxr=ivector(1,n);
      ipiv=ivector(1,n);
      for (j=1;j<=n;j++) ipiv[j]=0;
      for (i=1;i<=n;i++) {
            big=0.0;
            for (j=1;j<=n;j++)
                  if (ipiv[j] != 1)
                        for (k=1;k<=n;k++) {
                              if (ipiv[k] == 0) {
                                    if (fabs(a[j][k]) >= big) {
                                          big=fabs(a[j][k]);
                                          irow=j;
                                          icol=k;
                                    }
                              } else if (ipiv[k] > 1) {
                                    nrerror("GAUSSJ: Singular Matrix-1:
                                          Bad Initial Guesses");
                                    return;
                                    }
                        }
            ++(ipiv[icol]);
            if (irow != icol) {
                  for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
                  for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
            }
            indxr[i]=irow;
            indxc[i]=icol;
            if (a[icol][icol] == 0.0) {
                  nrerror("GAUSSJ: Singular Matrix-2:
                        Bad Initial Guesses");
                        return;
                        }
                pivinv=1.0/a[icol][icol];
                a[icol][icol]=1.0;
                for (l=1;l<=n;l++) a[icol][l] *= pivinv;
                for (l=1;l<=m;l++) b[icol][l] *= pivinv;
                for (ll=1;ll<=n;ll++)
                        if (ll != icol) {
                                dum=a[ll][icol];
                                a[ll][icol]=0.0;
                                for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
                                for (l=1;l<=m;l++) b[ll][l] -= b[icol][l]*dum;
                        }
        }
        for (l=n;l>=1;l--) {
                if (indxr[l] != indxc[l])
                        for (k=1;k<=n;k++)
                                SWAP(a[k][indxr[l]],a[k][indxc[l]]);
        }
        free_ivector(ipiv,1,n);
        free_ivector(indxr,1,n);
        free_ivector(indxc,1,n);
}

#undef SWAP
```

```
/*
        File Name:          LI.c
        Version:            1.0
        Project Name:       DataAcquisition
        Authors:            Program provided by National Instruments
        Compiler:           Lightspeed C 3.01

        Utility:            This program contains all the routines that can
                            be used with the National Instruments card

        Notes:              These routines are used to interface the Boxcar
*/

#include  <DeviceMgr.h>
#include  <stdio.h>
#include  <strings.h>
#include  <unix.h>

#include  "sys.h"
#include  "DrInterface.h"
#include  "GPIBres.h"

/*********************************************************/
int     ibbna(int,char*);               /*   d_id,boardname
        */
int     ibcac(int,int);                 /*   b_id,v                    */
int     ibclr(int);                     /*   d_id                      */
int     ibcmd(int,char*,long);          /*   b_id,buf,cnt              */
int     ibdma(int,int);                 /*   b_id,v                    */
int     ibeos(int,int);                 /*   db_id,v                   */
int     ibeot(int,int);                 /*   db_id,v                   */

int     ibfind(char*);                         /*   dev/board name
             */
int     ibgts(int,int);                 /*   b_id,v                    */
int     ibist(int,int);                 /*   b_id,v                    */
int     ibloc(int);                     /*   db_id                     */
int     ibonl(int,int);                 /*   db_id,v                   */
int     ibpad(int,int);                 /*   db_id,v                   */
int     ibpct(int);                     /*   d_id                      */
int     ibppc(int,int);                 /*   db_id,v                   */
int     ibrd(int,char*,long);           /*   db_id,buf,cnt             */
int     ibrda(int,char*,long);          /*   db_id,buf,cnt             */
int     ibrdf(int,char*);               /*   db_id,filename
        */
int     ibrpp(int,char*);               /*   db_id,buf                 */
int     ibrsc(int,int);                 /*   b_id,v                    */
int     ibrsp(int,char*);               /*   d_id,buf
        */
int     ibrsv(int,int);                 /*   d_id,v                    */
int     ibsad(int,int);                 /*   db_id,v                   */
int     ibsic(int);                     /*   b_id                      */
int     ibsre(int,int);                 /*   b_id,v                    */
int     ibstop(int);                    /*   db_id                     */
int     ibtmo(int,int);                 /*   db_id,v                   */
int     ibtrg(int);                     /*   d_id                      */
int     ibwait(int,int);                /*   db_id,mask                */
int     ibwrt(int,char*,long);          /*   db_id,buf,cnt             */
int     ibwrta(int,char*,long);         /*   db_id,buf,cnt              */
int     ibwrtf(int,char*);              /*   db_id,filename
        */


/***********************************************************************
***/
int     ibsta  =  0;
int     iberr  =  0;
int     ibret  =  -1;                   /* GPIB Status                  */
long    ibcnt  =  0L;                   /* variables                    */

int refNum  =  0x7FFF;                  /* ref. number of the driver    */

gpibBlock  paramBlk,  *paramBlkPtr;     /* param block for the driver
        */
StatusBlk  statusBlk;


/***********************************************************************
***/
int
ibfind(s)
char    *s;
{
        int  osErr;

        if  (OpenDriver(DRIVER_DNAME,  &refNum)){
               iberr  =  EDVR;
               ibsta  =  0x8000;
               return  -1;
        }

        paramBlkPtr  =  &paramBlk;
        paramBlkPtr->statusBlk  =  &statusBlk;
        paramBlkPtr->IOBufPtr  =  s;
        osErr  =  Control(refNum,  ibFIND,  &paramBlkPtr);
        if  (osErr){
               iberr  =  EDVR;
               ibsta  |=  U_ERR;
               return  -1;
        }
        else{
               ibsta  =  paramBlkPtr->statusBlk->ibsta;
               iberr  =  paramBlkPtr->statusBlk->iberr;
               ibret  =  paramBlkPtr->statusBlk->ibret;
               ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  paramBlkPtr->id;
}

/***********************************************************************
***/
int
ibrd      (id,  buf,  cnt)
int            id;
char    *buf;
long    cnt;
```

```
{

      OsErr  osErr;
      paramBlkPtr->id  =  id;
      paramBlkPtr->IOCount  =  cnt;
      paramBlkPtr->IOBufPtr  =  buf;
      osErr  =  Control(refNum,  ibRD,  &paramBlkPtr);
      if  (osErr  !=  0){
           iberr  =  EDVR;
           ibsta  |=  U_ERR;
      }
      else{
           ibsta  =  paramBlkPtr->statusBlk->ibsta;
           iberr  =  paramBlkPtr->statusBlk->iberr;
           ibret  =  paramBlkPtr->statusBlk->ibret;
           ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
      }
      return  ibsta;
}

/**********************************************************************
***/
int
ibwrt    (id,  buf,  cnt)
int      id;
char  *buf;
long  cnt;
{
      OsErr  osErr;

      paramBlkPtr->id  =  id;
      paramBlkPtr->IOCount  =  cnt;
      paramBlkPtr->IOBufPtr  =  buf;
      osErr  =  Control(refNum,  ibWRT,  &paramBlkPtr);
      if  (osErr  !=  0){
           iberr  =  EDVR;
           ibsta  |=  U_ERR;
      }
      else{
           ibsta  =  paramBlkPtr->statusBlk->ibsta;
           iberr  =  paramBlkPtr->statusBlk->iberr;
           ibret  =  paramBlkPtr->statusBlk->ibret;
           ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
      }
      return  ibsta;
}

/**********************************************************************
***/
int
ibcmd    (id,  buf,  cnt)
char  *buf;
long  cnt;
{
      OsErr  osErr;

      paramBlkPtr->id  =  id;
      paramBlkPtr->IOCount  =  cnt;
```

```
        paramBlkPtr->IOBufPtr  =  buf;
        osErr  =  Control(refNum,  ibCMD,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =   paramBlkPtr->statusBlk->ibsta;
                iberr   =   paramBlkPtr->statusBlk->iberr;
                ibret   =   paramBlkPtr->statusBlk->ibret;
                ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}

/********************************************************************
***/
int
ibtrg    (id)
int    id;
{
        OsErr  osErr;

        paramBlkPtr->id  =  id;
        osErr  =  Control(refNum,  ibTRG,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =   paramBlkPtr->statusBlk->ibsta;
                iberr   =   paramBlkPtr->statusBlk->iberr;
                ibret   =   paramBlkPtr->statusBlk->ibret;
                ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}

/********************************************************************
***/
int
ibclr(id)
int    id;
{
        OsErr  osErr;

        paramBlkPtr->id  =  id;
        osErr  =  Control(refNum,  ibCLR,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =   paramBlkPtr->statusBlk->ibsta;
                iberr   =   paramBlkPtr->statusBlk->iberr;
                ibret   =   paramBlkPtr->statusBlk->ibret;
                ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
        }
```

```
      return  ibsta;
}

/*********************************************************************
***/
int
ibpct    (id)
int   id;
{
      OsErr  osErr;

      paramBlkPtr->id  =  id;
      osErr  =  Control(refNum,  ibPCT,  &paramBlkPtr);
      if  (osErr  !=  0){
            iberr  =  EDVR;
            ibsta  |=  U_ERR;
      }
      else{
            ibsta   =   paramBlkPtr->statusBlk->ibsta;
            iberr   =   paramBlkPtr->statusBlk->iberr;
            ibret   =   paramBlkPtr->statusBlk->ibret;
            ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
      }
      return  ibsta;
}

/*********************************************************************
***/

ibllo (id)                          {
OsErr  osErr;

      paramBlkPtr->id  =  id;
      osErr  =  Control(refNum,  ibLLO,  &paramBlkPtr);
      if  (osErr  !=  0){
            iberr  =  EDVR;
            ibsta  |=  U_ERR;
      }
      else{
            ibsta   =   paramBlkPtr->statusBlk->ibsta;
            iberr   =   paramBlkPtr->statusBlk->iberr;
            ibret   =   paramBlkPtr->statusBlk->ibret;
            ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
      }
      return  ibsta;
}

/*********************************************************************
***/

ibloc    (id)                         {
OsErr  osErr;

      paramBlkPtr->id  =  id;
      osErr  =  Control(refNum,  ibLOC,  &paramBlkPtr);
      if  (osErr  !=  0){
            iberr  =  EDVR;
            ibsta  |=  U_ERR;
```

```
        }
        else{
                ibsta   =  paramBlkPtr->statusBlk->ibsta;
                iberr   =  paramBlkPtr->statusBlk->iberr;
                ibret   =  paramBlkPtr->statusBlk->ibret;
                ibcnt   =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}

/*********************************************************************
***/
int
ibrsp  (id,  spb)
int   id;
char  *spb;
{
        OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->IOBufPtr  =  spb;
        osErr  =  Control(refNum,  ibRSP,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =  paramBlkPtr->statusBlk->ibsta;
                iberr   =  paramBlkPtr->statusBlk->iberr;
                ibret   =  paramBlkPtr->statusBlk->ibret;
                ibcnt   =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}

/*********************************************************************
***/

ibrsv   (id, v)                        {
OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibRSV,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =  paramBlkPtr->statusBlk->ibsta;
                iberr   =  paramBlkPtr->statusBlk->iberr;
                ibret   =  paramBlkPtr->statusBlk->ibret;
                ibcnt   =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}
```

```
/*********************************************************************
***/
ibrpp   (id, buf) char *buf;        {
OsErr  osErr;

      paramBlkPtr->id  =  id;
      paramBlkPtr->IOBufPtr  =  buf;
      osErr  =  Control(refNum,  ibRPP,  &paramBlkPtr);
      if  (osErr  !=  0){
            iberr  =  EDVR;
            ibsta  |=  U_ERR;
      }
      else{
            ibsta  =  paramBlkPtr->statusBlk->ibsta;
            iberr  =  paramBlkPtr->statusBlk->iberr;
            ibret  =  paramBlkPtr->statusBlk->ibret;
            ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
      }
      return  ibsta;
}

/*********************************************************************
***/

ibsic   (id)                          {
OsErr  osErr;

      paramBlkPtr->id  =  id;
      osErr  =  Control(refNum,  ibSIC,  &paramBlkPtr);
      if  (osErr  !=  0){
            iberr  =  EDVR;
            ibsta  |=  U_ERR;
      }
      else{
            ibsta  =  paramBlkPtr->statusBlk->ibsta;
            iberr  =  paramBlkPtr->statusBlk->iberr;
            ibret  =  paramBlkPtr->statusBlk->ibret;
            ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
      }
      return  ibsta;
}

/*********************************************************************
***/

ibsre   (id, v)                  {
OsErr  osErr;

      paramBlkPtr->id  =  id;
      paramBlkPtr->controlVar  =  v;
      osErr  =  Control(refNum,  ibSRE,  &paramBlkPtr);
      if  (osErr  !=  0){
            iberr  =  EDVR;
            ibsta  |=  U_ERR;
      }
      else{
            ibsta  =  paramBlkPtr->statusBlk->ibsta;
            iberr  =  paramBlkPtr->statusBlk->iberr;
```

```
		ibret	=  paramBlkPtr->statusBlk->ibret;
		ibcnt	=  paramBlkPtr->statusBlk->ibcnt;
	}
	return  ibsta;
}

/***********************************************************************
***/

ibrsc   (id, v)                      {
OsErr  osErr;

	paramBlkPtr->id  =  id;
	paramBlkPtr->controlVar  =  v;
	osErr  =  Control(refNum,  ibRSC,  &paramBlkPtr);
	if  (osErr  !=  0){
		iberr  =  EDVR;
		ibsta  |=  U_ERR;
	}
	else{
		ibsta	=  paramBlkPtr->statusBlk->ibsta;
		iberr	=  paramBlkPtr->statusBlk->iberr;
		ibret	=  paramBlkPtr->statusBlk->ibret;
		ibcnt	=  paramBlkPtr->statusBlk->ibcnt;
	}
	return  ibsta;
}

/***********************************************************************
***/

ibist   (id, v)                     {
OsErr  osErr;

	paramBlkPtr->id  =  id;
	paramBlkPtr->controlVar  =  v;
	osErr  =  Control(refNum,  ibIST,  &paramBlkPtr);
	if  (osErr  !=  0){
		iberr  =  EDVR;
		ibsta  |=  U_ERR;
	}
	else{
		ibsta	=  paramBlkPtr->statusBlk->ibsta;
		iberr	=  paramBlkPtr->statusBlk->iberr;
		ibret	=  paramBlkPtr->statusBlk->ibret;
		ibcnt	=  paramBlkPtr->statusBlk->ibcnt;
	}
	return  ibsta;
}

/***********************************************************************
***/

ibonl    (id,  v){
OsErr  osErr;

	paramBlkPtr->id  =  id;
	paramBlkPtr->controlVar  =  v;
```

```
        osErr  =  Control(refNum,  ibONL,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =  paramBlkPtr->statusBlk->ibsta;
                iberr   =  paramBlkPtr->statusBlk->iberr;
                ibret   =  paramBlkPtr->statusBlk->ibret;
                ibcnt   =  paramBlkPtr->statusBlk->ibcnt;
        }
        if  (!v)
                CloseDriver(refNum);
        return  ibsta;
}

/**********************************************************************
***/

ibgts   (id, v)                         {
OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibGTS,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =  paramBlkPtr->statusBlk->ibsta;
                iberr   =  paramBlkPtr->statusBlk->iberr;
                ibret   =  paramBlkPtr->statusBlk->ibret;
                ibcnt   =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}

/**********************************************************************
***/

ibcac   (id, v)                         {
OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibCAC,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =  paramBlkPtr->statusBlk->ibsta;
                iberr   =  paramBlkPtr->statusBlk->iberr;
                ibret   =  paramBlkPtr->statusBlk->ibret;
                ibcnt   =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
```

```
}

/**********************************************************************
***/

ibppc   (id, v)                          {
OsErr  osErr;

       paramBlkPtr->id  =  id;
       paramBlkPtr->controlVar  =  v;
       osErr  =  Control(refNum,  ibPPC,  &paramBlkPtr);
       if (osErr != 0){
              iberr  =  EDVR;
              ibsta  |=  U_ERR;
       }
       else{
              ibsta  =  paramBlkPtr->statusBlk->ibsta;
              iberr  =  paramBlkPtr->statusBlk->iberr;
              ibret  =  paramBlkPtr->statusBlk->ibret;
              ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
       }
       return  ibsta;
}

/**********************************************************************
***/

ibdma(id,v){
OsErr  osErr;

       paramBlkPtr->id  =  id;
       paramBlkPtr->controlVar  =  v;
       osErr  =  Control(refNum,  ibDMA,  &paramBlkPtr);
       if (osErr != 0){
              iberr  =  EDVR;
              ibsta  |=  U_ERR;
       }
       else{
              ibsta  =  paramBlkPtr->statusBlk->ibsta;
              iberr  =  paramBlkPtr->statusBlk->iberr;
              ibret  =  paramBlkPtr->statusBlk->ibret;
              ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
       }
       return  ibsta;
}

/**********************************************************************
***/

ibwait   (id, v)                     {
OsErr  osErr;

       paramBlkPtr->id  =  id;
       paramBlkPtr->controlVar  =  v;
       osErr  =  Control(refNum,  ibWAIT,  &paramBlkPtr);
       if (osErr != 0){
              iberr  =  EDVR;
              ibsta  |=  U_ERR;
```

```
	}
	else{
		ibsta	=	paramBlkPtr->statusBlk->ibsta;
		iberr	=	paramBlkPtr->statusBlk->iberr;
		ibret	=	paramBlkPtr->statusBlk->ibret;
		ibcnt	=	paramBlkPtr->statusBlk->ibcnt;
	}
	return	ibsta;
}

/**********************************************************************
***/

boardstatus(id){
OsErr	osErr;

	paramBlkPtr->id	=	id;
	osErr	=	Control(refNum,	ibSTAT,	&paramBlkPtr);
	if	(osErr	!=	0){
		iberr	=	EDVR;
		ibsta	|=	U_ERR;
	}
	else{
		ibsta	=	paramBlkPtr->statusBlk->ibsta;
		iberr	=	paramBlkPtr->statusBlk->iberr;
		ibret	=	paramBlkPtr->statusBlk->ibret;
		ibcnt	=	paramBlkPtr->statusBlk->ibcnt;
	}
	return	ibsta;
}
/**********************************************************************
***/

ibpoke(id,v){
OsErr	osErr;

	paramBlkPtr->id	=	id;
	paramBlkPtr->controlVar	=	v;
	osErr	=	Control(refNum,	ibPOKE,	&paramBlkPtr);
	if	(osErr	!=	0){
		iberr	=	EDVR;
		ibsta	|=	U_ERR;
	}
	else{
		ibsta	=	paramBlkPtr->statusBlk->ibsta;
		iberr	=	paramBlkPtr->statusBlk->iberr;
		ibret	=	paramBlkPtr->statusBlk->ibret;
		ibcnt	=	paramBlkPtr->statusBlk->ibcnt;
	}
	return	ibsta;
}
/**********************************************************************
***/

ibeot	(id,	v)	{
OsErr	osErr;

	paramBlkPtr->id	=	id;
```

```
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibEOT,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =   paramBlkPtr->statusBlk->ibsta;
                iberr   =   paramBlkPtr->statusBlk->iberr;
                ibret   =   paramBlkPtr->statusBlk->ibret;
                ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}
ibpad    (id,  v)    {
OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibPAD,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =   paramBlkPtr->statusBlk->ibsta;
                iberr   =   paramBlkPtr->statusBlk->iberr;
                ibret   =   paramBlkPtr->statusBlk->ibret;
                ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}
ibsad    (id,  v)    {
OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibSAD,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
                ibsta  |=  U_ERR;
        }
        else{
                ibsta   =   paramBlkPtr->statusBlk->ibsta;
                iberr   =   paramBlkPtr->statusBlk->iberr;
                ibret   =   paramBlkPtr->statusBlk->ibret;
                ibcnt   =   paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}
ibeos    (id,  v)    {
OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibEOS,  &paramBlkPtr);
        if  (osErr  !=  0){
                iberr  =  EDVR;
```

```
               ibsta |= U_ERR;
        }
        else{
               ibsta  =  paramBlkPtr->statusBlk->ibsta;
               iberr  =  paramBlkPtr->statusBlk->iberr;
               ibret  =  paramBlkPtr->statusBlk->ibret;
               ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}
ibtmo    (id, v)   {
OsErr  osErr;

        paramBlkPtr->id  =  id;
        paramBlkPtr->controlVar  =  v;
        osErr  =  Control(refNum,  ibTMO,  &paramBlkPtr);
        if  (osErr  !=  0){
               iberr  =  EDVR;
               ibsta  |=  U_ERR;
        }
        else{
               ibsta  =  paramBlkPtr->statusBlk->ibsta;
               iberr  =  paramBlkPtr->statusBlk->iberr;
               ibret  =  paramBlkPtr->statusBlk->ibret;
               ibcnt  =  paramBlkPtr->statusBlk->ibcnt;
        }
        return  ibsta;
}

static  int  errno;/*
  *  file  I/O  routines:
  */

static  fserr()  {       /*  file  system  error  */
           iberr  =  EFSO;
           ibcnt  =  0L;
            return  ibsta  =  U_ERR  |  U_CMPL;
}


#define  RMODE  (O_RDONLY|O_EXCL)                   /*  read  only  */
#define  WMODE  (O_BINARY|O_WRONLY|O_CREAT)               /*  write  only  */

#define  FBSIZE   1024
static  char  buf[FBSIZE*2];

/*
  *  Write  data  to  GPIB  from  a  file.
  *  Eot  mode  is  disabled  until  the  last  block,  at  which  time,
  *  the  original  eot  mode  is  restored.   This  function  should  not
  *  be  called  unless  eot  mode  is  enabled  or  an  eos  byte  has
  *  been  established.
  *  Double  buffering  is  used  not  for  improved  thoughput,  but
  *  for  determining  the  last  buffer  before  it  is  sent.   This  allows
  *  the  eot  mode  to  be  restored  for  the  last  block  is  sent.
  */
ibwrtf(n,  file)  char  file[];  {
          register  i;
          unsigned long total=0L;
          int eot, osta, oerr;
             int vRef, ref;
             long cnt[2];

             GetVol(NULL, &vRef);

             CtoPstr(file);
           if ((FSOpen( file,vRef, &ref)) != 0)
                   return fserr();


         if (ibeot(n,0) & U_ERR)                          /* disable eot */
                 goto closf;
         eot = 1;                                /* save original eot mode */

         i = 0;
          cnt[0] = FBSIZE;
           if ((FSRead(ref, &cnt[0], buf)) != 0) {
                   errno = cnt[0];/*  fserr(); if higher level read */
                         /*       goto restor;            */          }
         while (cnt[i] > 0)  {                      i ^= 01;
                   cnt[i] = FBSIZE;
                     if ((FSRead(ref, &cnt[i], i?buf+FBSIZE:buf)) != 0){
                           errno = cnt[i]; /*      fserr();         */
                           /*      goto restor;    */                  }
                    if (cnt[i] <= 0)/* then the last read reached eof */
                            if (ibeot(n,1) & U_ERR)/* restore eot mode */
                                    goto closf;
                    if (ibwrt(n, i?buf:buf+FBSIZE,
                                (long)cnt[i^01]) & U_ERR) goto restor;
                  total += ibcnt;
         }
          goto closf;
restor:  osta = ibsta;
          oerr = iberr;
          ibeot(n,eot);                /* restore eot mode */
          ibsta = osta;
          iberr = oerr;
closf:   FSClose(ref);
           if (!(ibsta & U_ERR) || ((iberr != EDVR)&&(iberr != EFSO)))
                  ibcnt = total;
          return ibsta;
}

/*
       Read data from GPIB into a file until an END condition
       (eos or eot) is encountered.  Assumes that the sending device
       will assert EOI or that an EOS byte has been established.
 */

#define  fnfErr  -43
ibrdf(n,  file) char file[]; {
          unsigned long total=0L;
             int vRef, ref, err;

             GetVol(NULL, &vRef);
          CtoPstr(file);
         if ((err = FSOpen( file,vRef, &ref)) == fnfErr) {
        /* file not found (so create file) */
               if ((Create(file,vRef, '????', 'TEXT')) != 0)
                  return fserr();
              if ((FSOpen( file,vRef, &ref)) != 0)
                      return  fserr();
            }
         else if (err != 0)
          return  fserr();
         else
          SetEOF(ref,  0L);        /* clear existing file */

        do {
                if (ibrd(n,buf,(FBSIZE*2)) & U_ERR)
                     break;
                if ((FSWrite (ref, &ibcnt, &buf)) != 0) {
                     fserr();
                     break;
              }
              total += ibcnt;
         } while (!(ibsta & U_END));
        FSClose(ref);
         if (!(ibsta & U_ERR) ||((iberr != EDVR) && (iberr != EFSO)))
                ibcnt = total;
        return ibsta;
}
```

```
/*
	File Name:		MickeyMousefit.NLS.c
	Version:		1.0
	Project Name:		DataAcquisition
	Authors:		Mark J. Rosker, Marcos Dantus,
				Lawrence W. Peng
	Compiler:		Lightspeed C 3.01

	Utility:		This is the main portion of the NLS fitting
				routine.

	Notes:			This routine has been modified to include the
				selection of a portion of the data to be fitted.
				Look for additional notes
*/

*define		MAXFITPARAMS		20
/* Set to largest number of fit parameters		*/
*define		MAXNUMITER  50
*define		MINRELATIVEERROR  0.0000001
*define		T1		267.8
/* This is AC of pulse width, in steps		*/
*define		SQRTPI		1.772453851		/* sqrt of pi*/

*include	"nrutil.h"
*include	"MacTypes.h"
*include	"Data Acquisition.h"
*include	"math.h"

extern		int	NumOfDataPts;
extern		int		StartPos;
extern		int		EndPos;
extern		float XScale;
extern		Boolean FitParamOn[MAXFITPARAMS];
extern		float		*XData;
extern		float		**YData;
extern		int			NumYRows,  CurrYRow;
extern		Str255			JunkStr255;
extern		WindowPtr	PlotWindPtr;
extern		Boolean		FirstFit;
extern	int		CurrentFunction;
/*function being fit? Default to no function-LWP*/
extern	Boolean		FitWholeFile;
/*default to fit entire file-LWP*/

int		imode;

int	DoFitMenu(item) /* Taes care of menu selection for NLS fitting */
register	int		item;
{
	void	GaussianFunc();
	void	LorentzianFunc();
	void	HypSecantFunc();
	void	ExpStepFunc();
	void	AZmodelFunc();
	void	MJRmodelFunc();
	/*void		TestFunc();*/
	void		SingExpFunc();
```

```
        /*this plus other needed additions added LWP*/
        void     BiExpFunc();
        /*this plus other needed additions added LWP*/
        void     LineFunc();

        extern int  CountEm;

        DrawPlotWind(TRUE,XData,YData,NumOfDataPts,NumYRows,CurrYRow,NIL);
                    /* restore this!  */
        imode = item;                          /* ***very important!***
        */
        switch (item) {
              case ftGaussian:
                    NLSfit(GaussianFunc);
                    break;
              case ftLorentzian:
                    NLSfit(LorentzianFunc);
                    break;
              case ftHypSecant:
                    NLSfit(HypSecantFunc);
                    break;
              case ftExpStep:
                    NLSfit(ExpStepFunc);
                    break;
              case ftAZmodel:
                    NLSfit(AZmodelFunc);
                    break;
              case ftMJRmodel:
                    NLSfit(MJRmodelFunc);
                    break;
              case ftSingExp:
                    NLSfit(SingExpFunc);
                    break;
              case ftBiExp:
                    NLSfit(BiExpFunc);
                    break;
              case ftLine:
                    NLSfit(LineFunc);
                    break;
        /*    case ftTest:
                    NLSfit(TestFunc);
                    break;*/
              default:
                    break;
        }
}

NLSfit(TheFunc)
void  (*TheFunc)  ();

/*
After a great deal of effort, the fitting routine seems to be in work
ing order.
Changes made are the following:

First, all calls to the expression (*TheFunc) were changed to void.
This makes them consistent with the Numerical Recipes versions.
Initially, such calls were of the type double and the functions
```

declared as void. The program crashed when the routine DrawPlotWind() was called.

Second, all of the adjustable parameters, a() and dyda() used in the fitting functions have been changed to unit offset. That means that a(0) is a(1), dyda(0) is dyda(1), and so on. The above solution for the a[] parameters was the most direct way that I could think of to get the offsets to work with the Numerical Recipes routine. The Numerical recipes routine is geared for unit offsets.
Initially, I tried to achieve unit offset by using C commands within or within the fit parameter dialog box. Presumably, due to errors in the use of C or through some subtlety in the Numerical Reicpes routines, adjusting the offsets of any input parameters to mrqmin.c other than x, y, or sig gave a program crash, loss of computer control, infinite loops, or system bombs.

Okay, my methods may lack elegance....to hell with it.

Third, 'CurrentFunction' (set initially to zero) keeps track of which function you are using to fit the data. It is compared to the value of 'imode' most recently selected from the 'Fit Menu'. The result should be that if you stay with the same function, the dialog box will always display the most current parameter values (rather than always defaulting to the initial guesses). If you change the function, then 'CurrentFunction' no longer is equal to 'imode' and 'FirstFit' is reset to TRUE. Thus you get the initial guesses for the first run.

Fourth, free_vector and free_matrix statements have been inserted just before the final statement is this routine. It appears that the memory is "full" after 5-6 fits. I guess this works. I tried 20 fits, and it didn't crash on me yet. The error displayed is "error is dallocation". Previously, we never freed the vector declarations at the very beginning of 'NLS(TheFunc)'.

LOOK FOR OTHER SMALL NOTES IN THE ROUTINE!   LWP 10/23/88
*/

```
{
      static        double       a[MAXFITPARAMS];
      int           NumOfParams, NumOfParamsFit,  nca;
      double        chisq, alamda, LastChiSqr;
      double        MakeFakeData();
      char   StrFWHM[10];
      char   StrChiSqr[10];
      char   StrFitName[40];
      double        fwhm, factor;
      int           *lista;
      double        *x, *y, *sig, **covar, **alpha,*tempY;
      register      int          i,j, iter;
      int                 pos1[2],pos2[2],temp;
      float xstep;
      double        StartLog, EndLog, dummy[6];
      int                 MousePaint();
      int                 startend[2];
      void          nrerror();

      WindowPtr     TempPort;
```

```
    lista       =  ivector(1,MAXFITPARAMS);
    x           =  dvector(0,NumOfDataPts-1);
    y           =  dvector(0,NumOfDataPts-1);
    sig         =  dvector(0,NumOfDataPts-1);
    tempY       =  dvector(0,NumOfDataPts-1);
    covar =  dmatrix(1,MAXFITPARAMS,1,MAXFITPARAMS);
    alpha =  dmatrix(1,MAXFITPARAMS,1,MAXFITPARAMS);

    GetPort(&TempPort);                 /*    Save the current port
    */
    SetPort(PlotWindPtr);           /*    Point to plot window    */

    for (i = 0; i < NumOfDataPts; i++) {
          x[i] = XData[i];
          y[i] = YData[CurrYRow][i];
          sig[i] = 1.0;
    }

    /* Set up appropriate fit parameters              */
    switch (imode) {
          case ftGaussian: /* gaussian plus a constant.         */
                NumOfParams = 4;
                /* this is the total number of coefficients     */
                NumOfParamsFit    = 4;
                /* number of coefficients to fit (all)          */
                nca  = 4;         /* make this the same as
NumOfParams*/
                factor      = 1.6651;/* factor to go from a[3] to
fwhm*/
                break;
          case ftLorentzian:      /* Lorentzian plus a constant.
          */
                NumOfParams = NumOfParamsFit = nca = 4;
                factor      = 2.0;/* factor to go from a[3] to fwhm
    */
                break;
          case ftHypSecant:
          /* Hyperbolic secant squared plus a constant.   */
                NumOfParams = NumOfParamsFit = nca = 4;
                factor      = 1.7627;/* factor to go from a[3] to
fwhm*/
                break;
          case ftExpStep:
                NumOfParams = NumOfParamsFit = nca = 4;
                factor      = 1.7627;/* factor to go from a[3] to
fwhm*/
                break;
          case ftAZmodel:
                NumOfParams = NumOfParamsFit = nca = 6;
                factor      = 1.0;
                /* factor not meaningful for this case*/
                break;
          case ftMJRmodel  :
                NumOfParams = NumOfParamsFit = nca   = 5;
                factor = 1.0;/* factor not meaningful for this case
    */
                break;
          case ftSingExp:
```

```
				NumOfParams = NumOfParamsFit = nca = 4;
				factor = 1.0;	/*factor not meaningful in this case
	*/
				break;
			case ftBiExp:
				NumOfParams = NumOfParamsFit = nca = 6;
				factor = 1.0;	/*factor not meaningful in this case
	*/
				break;
			case ftLine:
				NumOfParams = NumOfParamsFit = nca = 2;
				factor = 1.0;	/*factor not meaningful in this case
	*/
				break;
			case ftTest :
				NumOfParams = NumOfParamsFit = nca = 5;
				factor     = 1.7627;/* factor to go from a[3] to
fwhm*/					break;
			default:
				DebugStr("NLSfit: Hit the default");
				break;
	}
	if (CurrentFunction != imode) {
			FirstFit = TRUE;
			CurrentFunction = imode;
			}
			else{
			FirstFit = FALSE;}

	if (FirstFit) {
			GetInitialGuesses(x,y,a,NumOfDataPts,factor);
			for (i = 0,j=0; i < NumOfParams; i++) {
				FitParamOn[i] = TRUE;
			}
	}

	ParamText("\pEnter initial guesses for fit parameters:",
			"\p","\p","\p");

	 if(!DoFitDlog(a,FitParamOn,NumOfParams)){
	goto AllDone;}
	 DrawPlotWind(TRUE,XData,YData,NumOfDataPts,NumYRows,
	CurrYRow,NIL);


	if(!FitWholeFile) {
			MousePaint(&startend);
			pos1[1] = startend[1];
			pos2[1] = startend[2];

			xstep  =XData[1]-XData[0];
			StartPos = (int) (((pos1[1] - 50) * XScale)/xstep);
			EndPos = (int) (((pos2[1] - 50) * XScale)/xstep);

			if(StartPos < 0) StartPos = 0;
			if(EndPos > NumOfDataPts) EndPos = NumOfDataPts;

	/*
```

```
For the moment, if you want to fit single exponentials, must
select part of a file-LWP
*/
      if((StartPos != 0 || EndPos != NumOfDataPts) &&
            ( imode == ftSingExp || imode== ftLine))
            {
            if(imode==ftSingExp)
                  {
                  a[2] = XData[StartPos];
                  if(YData[CurrYRow][StartPos] <= 0.0)
                        StartLog = YData[CurrYRow][StartPos];
                  else StartLog =log(YData[CurrYRow][StartPos]);

                  if(YData[CurrYRow][EndPos] <= 0.0)
                        EndLog = YData[CurrYRow][EndPos];
                  else EndLog =log(YData[CurrYRow][EndPos]);

            a[3] = -(XData[EndPos]-XData[StartPos])
                  /(EndLog-StartLog);
                  a[1] = exp(StartLog + (XData[StartPos]-a[2])
                        /a[3]);
                  a[4] = 0.0;
            }
            else
            {
                  if(FitParamOn[0])
                  {
                        a[1] = (YData[CurrYRow][EndPos]
                              -YData[CurrYRow][StartPos])
                              /(XData[EndPos]
                              - XData[StartPos]);
                  }
                  if(FitParamOn[1])
                  {
                        a[2] = ( YData[CurrYRow][StartPos]
                              + YData[CurrYRow][EndPos] -a[1]
                              * (XData[StartPos]
                              + XData[EndPos]))/2.0;
                  }
            }
      }

}
else
{
      StartPos = 0;
      EndPos = NumOfDataPts;
}


for (i = 0, j = 0, NumOfParamsFit = 0; i < NumOfParams; i++)
{     /* which params should be fit?      */
      if (FitParamOn[i]) {
            lista[++j] = i+1; /* select params to be varied */
            NumOfParamsFit++;
      }
}
```

```
	alamda = -1.0;
	chisq = 1.0E400;  /* a real big number*/
	iter = 0;

/*
A few notes about the displayed fit. For each iteration through the
do loop, the window displays the current best fit and current
iteration number. The DrawPlotWind routine destroys displayed text
about the previous iteration. Thus the DrawString for displaying the
iteration number is put into the do loop.  As an alternative, one
could comment out the DrawPlotWind and move the next three lines out
just before the do loop. This has the effect of displaying the
iteration numbers sequentially on the screen (as each iteration
starts), but does not display the final fit until the next call to
DrawPlotWind (at the end of the NLS(TheFunc) subroutine).
This alternative display method is actually used in the current
'Data Display' application of 047. It is your choice. In the
above do-while loop, if you do not comment out 'DrawPlotWind', the
fit for all values of x will be plotted.  Otherwise, the fit for
the portion of the data fitted is displayed.   LWP    10/22/88
*/

	do
	{
		/*DrawPlotWind(TRUE,XData,YData,NumOfDataPts,NumYRows,
			CurrYRow,TheFunc,a,NumOfParams);*/
		/*above statement permits showing a partial file fit*/
		MoveTo(50,25);
		ForeColor(magentaColor);
		DrawString("\pIteration  #");
		MoveTo(120+10*iter,25);
		sprintf(JunkStr255,"%d",iter + 1);
		CtoPstr(JunkStr255);
		DrawString(JunkStr255);
		LastChiSqr = chisq;
		mrqmin(x-1,y-1,sig-1,EndPos,a,NumOfParams,lista,
		NumOfParamsFit,covar,alpha,&chisq,TheFunc,&alamda);
		if (chisq == 0.0) break;
		/* special check -- real good data! */
		if (++iter >= MAXNUMITER)
		{
			nrerror("Maximum number of fit iterations exceeded");
			break;
		}
	} while ((LastChiSqr - chisq)/LastChiSqr > MINRELATIVEERROR);

	switch (imode) {
		case ftGaussian:
			strcpy(StrFitName,"Fit to Gaussian:");
			break;
		case ftLorentzian:
			strcpy(StrFitName,"Fit to Lorentzian:");
			break;
		case ftHypSecant:
			strcpy(StrFitName,"Fit to Hyperbolic secant
squared:");
			break;
		case ftExpStep:
```

```
                    strcpy(StrFitName,"Fit  to  Exponential  Step:");
                    break;
               case  ftAZmodel:
                    strcpy(StrFitName,"Fit  to  B&Z's  model:");
                    break;
               case  ftMJRmodel:
                    strcpy(StrFitName,"Fit  to  MJR's  model:");
                    break;
               case  ftSingExp:
                    strcpy(StrFitName,"Fit  to  Single  Exponential  model:");
                    break;
               case  ftBiExp:
                    strcpy(StrFitName,"Fit  to  BiExponential  model:");
                    break;
               case  ftLine:
                    strcpy(StrFitName,"Fit  to  Linear  model:");
                    break;
               case  ftTest:
                    strcpy(StrFitName,"Fit  to  Test  model:");
                    break;
          }
          fwhm  =  a[3]  *  factor;

          sprintf(StrFWHM,"%.3G",fwhm);
          sprintf(StrChiSqr,"%.3G",chisq);
          CtoPstr(StrFitName);
          CtoPstr(StrFWHM);
          CtoPstr(StrChiSqr);
          ParamText(StrFitName,StrFWHM,StrChiSqr,"\p");
          if  (!DoFitDlog(a,FitParamOn,NumOfParams))  goto  AllDone;
          DrawFit(x,  a,  NumOfParams,TheFunc);

          /*DrawPlotWind(TRUE,XData,YData,NumOfDataPts,NumYRows,CurrYRow,
               TheFunc,  a,  NumOfParams);*/
               for  (i  =  StartPos;  i  <  EndPos;  i++)
               {       /*  now  save  the  fit  */
                    (*TheFunc)  (x[i],a,&tempY[i],dummy,NumOfParams);
                    sig[i]  =  YData[CurrYRow][i];
                    /*  borrow  the  sig  array  as  temporary  storage  */
               }
          for  (i  =  StartPos;  i  <  EndPos;  i++)
               YData[CurrYRow][i]=tempY[i];
          WriteData(XData,YData,NumOfDataPts,NumYRows,CurrYRow);
          for  (i  =  StartPos;  i  <  EndPos;  i++)
               YData[CurrYRow][i]=sig[i];
/*
The  above  statement  used  to  be  'DrawPlotWind'.  However,  this  gave
nothing  but  zeros  when  one  tried  to  save  the  fit  into  a  new  file.  LWP
*/

          DrawFit(x,  a,  NumOfParams,TheFunc);

/*
For  some  strange  reason,  the  dialog  box  for  saving/naming  data  files
produced  by  the  'WriteData()'  sometimes  erases  any  part  of  the  fit
which  it  covers.   With  any  luck,  a  second  call  of  'DrawFit()'  will
solve  this  "random"  problem.   LWP
*/
```

```
AllDone:
      free_dvector(x,0,NumOfDataPts-1);
      /*This group of 6 statements frees the memory*/
      free_dvector(y,0,NumOfDataPts-1);
      /*allocated for each run of NLS(TheFunc)*/
      free_dvector(sig,0,NumOfDataPts-1);
      free_ivector(lista,1,MAXFITPARAMS);
      free_dvector(tempY,0,NumOfDataPts-1);
      free_dmatrix(covar,1,MAXFITPARAMS,1,MAXFITPARAMS);
      free_dmatrix(alpha,1,MAXFITPARAMS,1,MAXFITPARAMS);

      SetPort(TempPort);                    /*    Return to old window
      */
}

DrawFit(x, a, NumOfParams, TheFunc)
int         NumOfParams;
double      x[], a[];
void  (*TheFunc) ();
{
      double      fity;
      int         i;
      Point APoint;
      double      dyda[MAXFITPARAMS];      /* this is really just junk
      */
      ForeColor(cyanColor);
      for (i = StartPos; i < EndPos; i++) {
            (*TheFunc)  (x[i],a,&fity,dyda,NumOfParams);
            GetPointFromData((float) x[i],(float) fity,&APoint);

            if (i == StartPos)
                  MoveTo(APoint.h,APoint.v);
            else
                  LineTo(APoint.h,APoint.v);
      }
}

GetInitialGuesses(x,y,a,npts,ScaleFullWidth)
/* puts initial guesses for parameters into a*/
int               npts;
double            x[],y[],a[], ScaleFullWidth;
{

      double      baseline();

      double      xmax,ymax,xmin,ymin,yh, xl, xr;
      register    int   i;
      int   counter = 0;
      double      blsuml = 0.0, blsumr = 0.0;

      switch (imode) {
            case  ftGaussian:
            case  ftLorentzian:
            case  ftHypSecant:

                  a[4]  =     baseline(x,y,npts);
```

```
        FindMinMaxY(x,y,npts,&xmax,&ymax,&xmin,&ymin);

        if  (fabs(ymax)  >=  fabs(ymin))
        {           /* positive maximum  */
              a[1]  = ymax - a[4];
              a[2]  = xmax;
              yh = (a[1] + a[4])/2.0;

              i = 0;
              while (y[i] < yh)
              {                  /* get left halfpoint */
                    i++;
              }
              xl = x[i];

              i = npts - 1;
              while (y[i] < yh)
              {                  /* get right halfpoint */
                    i--;
              }
              xr = x[i];
        } else {
              a[1]     = ymin - a[4];
          a[2]       = xmin;
              yh = (a[1] + a[4])/2.0;

              i = 0;
              while (y[i] > yh)
              {                  /* get left halfpoint */
                    i++;
              }
              xl = x[i];

              i = npts - 1;
              while (y[i] > yh)
              {                  /* get right halfpoint */
                    i--;
              }
              xr = x[i];
        }
        a[3]  =  fabs(xr-xl)/ScaleFullWidth;
        break;
  case  ftSingExp:
        FindMinMaxY(x,y,npts,&xmax,&ymax,&xmin,&ymin);
              a[2] = xmax;
        a[3]  =  -(XData[NumOfDataPts-1]-XData[0])
              /(log(YData[CurrYRow][NumOfDataPts-1])
              -log(YData[CurrYRow][0]));
              a[1]  =   exp(log(YData[CurrYRow][0])
                    +  (XData[0]-a[2])/a[3]);
              a[4]  = ymin;
        break;
  case  ftLine:
        a[1]  =  (YData[CurrYRow][NumOfDataPts-1]
              -  YData[CurrYRow][0])/(XData[NumOfDataPts-1]
              -  XData[0]);
        a[2] = YData[CurrYRow][0] - a[1] * XData[0];
        break;
```

```
            case  ftBiExp:
                  FindMinMaxY(x,y,npts,&xmax,&ymax,&xmin,&ymin);
                  a[1] = ymax * 2.0;
                  a[2] = 100.0;
                  a[3] = 100.0;
                  a[4] = xmax;
                  a[5] = -1.0;
                  a[6] = ymin;
                  break;
            case  ftExpStep:
                  a[4]  =  baseline(x,y,npts);
                  a[3] = 300.0;
                  a[1] = -1.7;
                  a[2] = 5700.0;
                  break;
            }
}

double      baseline(x,y,npts)
double      x[],y[];
int         npts;
{
      register    int   i;
      int   counter = 0;
      double      blsum = 0.0;

      i = 0;                                        /* add left side  */
      while (i < npts/10)
      {
            blsum += y[i++];
            counter++;
      }

      if (imode != ftBiExp)
      {
            i = npts * 9 / 10;                      /*add right side*/
            while (i < npts)
            {
                  blsum += y[i++];
                  counter++;
            }
      }

      return(blsum/ counter);
}
FindMinMaxY(x,y,npts,xmaxPtr,ymaxPtr,xminPtr,yminPtr)
/*the x correspond to min, max of y*/
int               npts;
double            x[],y[];
double            *xmaxPtr;
double            *ymaxPtr;
double            *xminPtr;
double            *yminPtr;

{
      register    int   i;

      *ymaxPtr = y[0];
```

```
        *xmaxPtr  =  x[0];
        *yminPtr  =  y[0];
        *xminPtr  =  x[0];

        for (i = 1; i < npts; i++) {
                if (y[i] >= *ymaxPtr) {
                        *ymaxPtr  =  y[i];
                        *xmaxPtr  =  x[i];
                }
                if (y[i] <= *yminPtr) {
                        *yminPtr  =  y[i];
                        *xminPtr  =  x[i];
                }
        }
}
```

```
/*
      File Name:          mini.windows.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Program part of the LightSpeedC MiniEdit project
      Compiler:           Lightspeed C 3.01

      Utility:            This program contains the routines that control
                          the windows of the graphic interface

      Notes:              window functions for Miniedit,
                          addapted by Mark J. Rosker
*/

#include  "QuickDraw.h"
#include  "MacTypes.h"
#include  "WindowMgr.h"
#include  "TextEdit.h"
#include  "ControlMgr.h"
#include  "EventMgr.h"

#include  "Data Acquisition.h"

extern      WindowRecord       CommWindRec,EditWindRec,PlotWindRec;
extern      WindowPtr          CommWindPtr,EditWindPtr,PlotWindPtr;
extern  ControlHandle  vScroll;
extern  TEHandle    TEH;
extern  char              dirty;
extern  Str255            theFileName;
extern  int               linesInFolder;

Point theOrigin;

SetUpWindows()
{
      Rect  destRect,  viewRect;
      Rect  vScrollRect;
      FontInfo    myInfo;
      int         height;

      SetPort((CommWindPtr  =  GetNewWindow(  COMMWINDOWID,
                                              &CommWindRec,-1L  )));
      (*(WindowPeek)CommWindPtr).windowKind  =  COMMWINDKIND;/*     added
mjr*/

      TextFont(4);
      TextSize(9);
      vScrollRect  =  (*CommWindPtr).portRect;
      vScrollRect.left  =  vScrollRect.right-15;
      vScrollRect.right  +=  1;
      vScrollRect.bottom  -=  14;
      vScrollRect.top  -=  1;
      vScroll  =  NewControl(  CommWindPtr,  &vScrollRect,
                                "\p",  1,  0,  0,  0,scrollBarProc,  0L);
      viewRect  =  (*thePort).portRect;
      viewRect.right  -=  SBarWidth;
      viewRect.bottom  -=  SBarWidth;
      destRect  =  viewRect;
```

```
	InsetRect( &destRect, 4, 0 );
	GetFontInfo( &myInfo );
	height = myInfo.ascent+myInfo.descent+myInfo.leading;
	linesInFolder =
		(viewRect.bottom-viewRect.top)/height;
	viewRect.bottom = linesInFolder*height;
	TEH = TENew( &destRect, &viewRect );
	SetPt( &theOrigin, 0, 0 );
	dirty = 0;
}

CloseCommWindPtr()
{
	HideWindow( CommWindPtr );
	TESetSelect( 0, (**TEH).teLength, TEH );
	TEDelete( TEH );
	SetVScroll();
	SetUpFiles();
}

UpdateWindow(theWindow)
WindowPtr	theWindow;
{
	GrafPtr		savePort;

/*	GetPort( &savePort );				modified MJR
	SetPort( theWindow );	*/
	SetVScroll(TEH);
	ScrollBits(theWindow);
/*	BeginUpdate( theWindow );	*/
	DrawGrowIcon( theWindow );
	TEUpdate( &(**TEH).viewRect, TEH );
	DrawControls( theWindow );
/*	EndUpdate( theWindow );
	SetPort( savePort );		*/
}

ScrollBits(theWindow)
WindowPtr	theWindow;
{
	Point		oldOrigin;
	RgnHandle	tmpRgn;
	int			dv;

	oldOrigin = theOrigin;
	theOrigin.v = (**TEH).lineHeight*GetCtlValue(vScroll);
	dv = oldOrigin.v - theOrigin.v;

	OffsetRect(&(**TEH).destRect, 0, dv);
	tmpRgn=NewRgn();
	if (dv!=0) {
		ScrollRect( &(**TEH).viewRect, 0, dv, tmpRgn );
		InvalRgn( tmpRgn );
	}
	DisposeRgn( tmpRgn );
}

SetVScroll()
```

```
{
	register int max, n;

	n=(**TEH).nLines-linesInFolder;
	if ( ((**TEH).teLength>0) &&
		( (*((**TEH).hText))[(**TEH).teLength-1]=='\r' )) n += 1;
	max = ( n>0 ? n : 0 );
		SetCtlMax( vScroll, max );
}

int   scrollCode;
int  scrollAmt;

pascal void ScrollProc(theControl, theCode)
ControlHandle     theControl;
int                         theCode;
{
	int locVal;
	GrafPtr theWindow;

	if (theCode==scrollCode) {
		locVal = GetCtlValue(theControl);
		GetPort( &theWindow );
		SetVScroll(TEH);
		SetCtlValue( theControl, locVal+scrollAmt );
		UpdateWindow(CommWindPtr);
	}
}

DoContent(theWindow, theEvent)
WindowPtr   theWindow;
EventRecord *theEvent;
{
	int                         cntlCode;
	ControlHandle      theControl;
	int                         pageSize;
	GrafPtr                     savePort;

	GetPort(&savePort);
	SetPort(theWindow);
	GlobalToLocal( &(*theEvent).where );
	switch ( cntlCode = FindControl(
		(*theEvent).where, theWindow, &theControl ) ) {
	case inUpButton:
	case inDownButton:
	case inPageUp:
	case inPageDown:
		if (theControl==vScroll) {
			pageSize = ((**TEH).viewRect.bottom -
				(**TEH).viewRect.top) / (**TEH).lineHeight - 1;
			switch (cntlCode) {
			case inUpButton:
				scrollAmt = -1;
				break;
			case inDownButton:
				scrollAmt = 1;
				break;
			case inPageUp:
```

```
                        scrollAmt  =  -pageSize;
                        break;
                   case  inPageDown:
                        scrollAmt  =  pageSize;
                        break;
                   }
                   scrollCode  =  cntlCode;

        if(TrackControl(theControl,(*theEvent).where,&ScrollProc));
              }
              break;
        case  inThumb:
              if  (TrackControl(theControl,  (*theEvent).where,  0L  ))  ;
              UpdateWindow(theWindow);
              break;
        default:
              if  (PtInRect(  (*theEvent).where,  &(**TEH).viewRect  ))
                   TEClick(  (*theEvent).where,
                        (((*theEvent).modifiers  &  shiftKey  )!=0),  TEH);
        }
        SetPort(savePort);
}

MyGrowWindow(  w,  p  )
WindowPtr  w;
Point  p;
{
        GrafPtr     savePort;
        Rect  oldRect,  tempRect;
        long  theResult;
        Point thePt;
        Rect   r;

        GetPort(  &savePort  );
        SetPort(  w  );
        oldRect  =  (*w).portRect;
        SetRect(&tempRect,  80,  80,  screenBits.bounds.right,
                   screenBits.bounds.bottom);
        theResult  =  GrowWindow(  w,  p,  &tempRect  );
        thePt.h  =  LoWord(  theResult  );
        thePt.v  =  HiWord(  theResult  );
        SizeWindow(  w,  thePt.h,  thePt.v,  1  );
        InvalRect(  &(*w).portRect  );
        MoveControl(  vScroll,  (*w).portRect.right-SBarWidth,
              (*w).portRect.top-1);
        SizeControl(  vScroll,  SBarWidth+1,
              (*w).portRect.bottom-(*w).portRect.top-(SBarWidth-2));
        (**TEH).viewRect  =  (*w).portRect;
        (**TEH).viewRect.right  -=  SBarWidth+1;
        (**TEH).viewRect.bottom  -=  SBarWidth+1;
        (**TEH).destRect.right  =  (**TEH).viewRect.right;
        linesInFolder  =
              ((**TEH).viewRect.bottom-(**TEH).viewRect.top)
                   /(**TEH).lineHeight;
        (**TEH).viewRect.bottom  =  (**TEH).viewRect.top  +
                        (**TEH).lineHeight*linesInFolder;
        EraseRect(&(*w).portRect);
        r  =  (*w).portRect;
```

```
	r.top = r.bottom - (SBarWidth+1);
	InvalRect(&r);
	TECalText( TEH );
	UpdateWindow(w);
	SetPort( savePort );
}

ShowSelect()
{
	register int		theLine, point;
	int			curLine, adjust;

	SetVScroll();
	curLine = GetCtlValue( vScroll );
	point = (**TEH).selStart;
	for (theLine=0;
		((point<(**TEH).lineStarts[theLine]) ||
		(point>=(**TEH).lineStarts[theLine+1]))
		; theLine++ ) ;
	if ( (theLine<curLine ) || (theLine>(curLine+linesInFolder))) {
		adjust = (linesInFolder-1)/2;
		SetCtlValue( vScroll, theLine-adjust );
		}
	UpdateWindow(CommWindPtr);
}
```

```
/*
      File Name:          MousePaint.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Marcos Dantus, Lawrence W. Peng
      Compiler:           Lightspeed C 3.01

      Utility:            This file allows selection of a portion of
                          data to be fitted by selecting with the
                          mouse the starting and ending points
*/

MousePaint(startend)
int   startend[];
{
      int         pos1[2],pos2[2],pos3[2],pos4[2];
      int         *redRect,*theRect;
      int          temp;

            PenMode(srcOr);

            PenPat(ltGray);
            ForeColor(blueColor);

            while(!Button());
            GetMouse(&pos1);
            /* Gets the first point to start the fit */


            GetMouse(&pos3);
            SetRect(&theRect,pos1[1],31,pos3[1]+1,318);
            PaintRect(&theRect);

            while(Button()){
                  GetMouse(&pos4);
                  if(pos3[1] > pos4[1]){
                        ForeColor(whiteColor);
                        PaintRect(&theRect);
                        ForeColor(blueColor);
                  }
                  SetRect(&theRect,pos1[1],31,pos4[1],318);
                  PaintRect(&theRect);
                  pos3[1] = pos4[1];
            }
            GetMouse(&pos2);
            /* Gets the second point to end the fit */
            if(   pos1[1] > pos2[1]) {
                  temp = pos2[1];
                  pos2[1] = pos1[1];
                  pos1[1] = temp;
            }
            ForeColor(whiteColor);
            PaintRect(&theRect);
            PenMode(srcCopy);
            ForeColor(cyanColor);
            PenPat(black);
            startend[1] = pos1[1]; startend[2] = pos2[1];
}
```

```
/*
        File Name:          mrqcof.c
        Version:            1.0
        Project Name:       DataAcquisition
        Authors:            subroutine from Numerical Recipes
        Compiler:           Lightspeed C 3.01

        Utility:            This file is used by mrqmin.c to evaluate
                            the linearized fitting matrix **alpha,
                            and vector beta[]

        Note:               In this modified version, double is used
                            instead of float
*/

extern      int StartPos;

void  mrqcof(x,y,sig,ndata,a,ma,lista,mfit,alpha,beta,chisq,funcs)
double  x[],y[],sig[],a[],**alpha,beta[],*chisq;
int  ndata,ma,lista[],mfit;
void  (*funcs)();
{
        int  k,j,i;
        double  ymod,wt,sig2i,dy,*dyda,*dvector();
        void  free_dvector();

        dyda=dvector(1,ma);
        for  (j=1;j<=mfit;j++)  {
                for  (k=1;k<=j;k++)  alpha[j][k]=0.0;
                beta[j]=0.0;
        }
        *chisq=0.0;
        for  (i=StartPos+1;i<=ndata;i++)  {
                (*funcs)(x[i],a,&ymod,dyda,ma);
                /*printf("y=%f\n",ymod);*/
                sig2i=1.0/(sig[i]*sig[i]);
                dy=y[i]-ymod;
                for  (j=1;j<=mfit;j++)  {
                        wt=dyda[lista[j]]*sig2i;
                        for  (k=1;k<=j;k++)
                                alpha[j][k]  +=  wt*dyda[lista[k]];
                        beta[j]  +=  dy*wt;
                }
                (*chisq)  +=  dy*dy*sig2i;
        }
        for  (j=2;j<=mfit;j++)
                for  (k=1;k<=j-1;k++)  alpha[k][j]=alpha[j][k];
        free_dvector(dyda,1,ma);
}
```

```
/*
	File Name:		mrqmin.c
	Version:		1.0
	Project Name:		DataAcquisition
	Authors:		subroutine from Numerical Recipes
	Compiler:		Lightspeed C 3.01

	Utility:		Levenberg-Marquardt method, attempting to reduce
				the value of chisq of a fit between a set of
				ndata points x[], y[] with individual standard
				deviations sig[], and a nonlinear function
				dependent on ma coefficients a[].

	Note:			In this modified version, double is used
				instead of float
*/

void	mrqmin(x,y,sig,ndata,a,ma,lista,mfit,covar,alpha,chisq,
	funcs,alamda)
double	x[],y[],sig[],a[],**covar,**alpha,*chisq,*alamda;
int	ndata,ma,lista[],mfit;
void	(*funcs)();
{
	int	k,kk,j,ihit;
	static	double	*da,*atry,**oneda,*beta,ochisq;
	double	*dvector(),**dmatrix();
	void	mrqcof(),gaussj(),covsrt(),nrerror(),free_dmatrix(),
		free_dvector();

	if (*alamda < 0.0) {
		oneda=dmatrix(1,mfit,1,1);
		atry=dvector(1,ma);
		da=dvector(1,ma);
		beta=dvector(1,ma);
		kk=mfit+1;
		for (j=1;j<=ma;j++) {
			ihit=0;
			for (k=1;k<=mfit;k++)
				if (lista[k] == j) ihit++;
			if (ihit == 0)
				lista[kk++]=j;
			else
			if (ihit > 1) nrerror("Bad LISTA permutation in
MRQMIN-1");
		}
		if (kk != ma+1) nrerror("Bad LISTA permutation in MRQMIN-
2");
		*alamda=.1; /*Originally .001*/

	mrqcof(x,y,sig,ndata,a,ma,lista,mfit,alpha,beta,chisq,funcs);
		ochisq=(*chisq);
	}
	for (j=1;j<=mfit;j++) {
		for (k=1;k<=mfit;k++) covar[j][k]=alpha[j][k];
		covar[j][j]=alpha[j][j]*(1.0+(*alamda));
		oneda[j][1]=beta[j];
	}
	gaussj(covar,mfit,oneda,1);
```

```
        for  (j=1;j<=mfit;j++)
                da[j]=oneda[j][1];
        if (*alamda == 0.0) {
                covsrt(covar,ma,lista,mfit);
                free_dvector(beta,1,ma);
                free_dvector(da,1,ma);
                free_dvector(atry,1,ma);
                free_dmatrix(oneda,1,mfit,1,1);
                return;
        }
        for  (j=1;j<=ma;j++)  atry[j]=a[j];
        for  (j=1;j<=mfit;j++)
                atry[lista[j]]  =  a[lista[j]]+da[j];
        mrqcof(x,y,sig,ndata,atry,ma,lista,mfit,covar,da,chisq,funcs);
        if (*chisq < ochisq) {
                *alamda *= 0.2; /* originally 0.1 */
                ochisq=(*chisq);
                for  (j=1;j<=mfit;j++)  {
                        for  (k=1;k<=mfit;k++)  alpha[j][k]=covar[j][k];
                        beta[j]=da[j];
                        a[lista[j]]=atry[lista[j]];
                }
        } else {
                *alamda *= 10.0;
                *chisq=ochisq;
        }
        return;
}
```

```
/*
      File Name:         NewBoxcar.c
      Version:           1.0
      Project Name:      DataAcquisition
      Authors:           Mark J. Rosker
      Compiler:          Lightspeed C 3.01

      Utility:           this file contains EVERYTHING necessary for
                         boxcar communication through the IEEE (GPIB)
                         port.  Uses calls to high-level National
                         Instruments subroutines. (in the file LI.c)

      Notes:             Note the static variables.
                         Modified by Marcos Dantus Sept '89
*/

#include  "TextEdit.h"
#include  "IEEE.h"

#include  "Data Acquisition.h"
#define      BADVOLTAGE 11.00 /* a badvoltage is more than +/- 10 v
      */

#define      OUTBOXBUFFERLENGTH      255

extern       PortConfig  boxcarPort;
static       int               boxcarDVM;         /* IEEE device number
             */
static       char        outBoxcarBuff[OUTBOXBUFFERLENGTH];
                         /* output buffer sent to boxcar (keyboard)
      */
static       int               outBoxcarBuffIndex;

extern       char  JunkCharArray[255];

float GetBoxcarVoltage(BoxcarChannel)
                         /* returns the voltage of a boxcar channel
      */
int          BoxcarChannel;
{
      Str255             AStr255;
      float       Voltage;
      float       atof();

top:
      BoxcarInquire(BoxcarChannel);
             /* Inquire what the voltage is on a boxcar ADC channel
      */

      if (ibrd (boxcarDVM,AStr255,255L) & ERR)
                                        BoxcarError("Bad read in
GetBoxcarVoltage");
      CleanUpString(AStr255);
      sscanf(AStr255,"%f",&Voltage);

      if (Voltage > 10.237 || Voltage < -10.237) {
             sprintf(JunkCharArray,
             "Got a bad voltage from the boxcar: %f volts",Voltage);
```

```
			BoxcarError(JunkCharArray);
			goto top;
		}
		return(Voltage);
}

CleanUpString(AStr255)		/* assumes a C String	*/
Str255		AStr255;
{
		int		i,length;
		char	*TheCharPtr;

		length = strlen(AStr255);
		TheCharPtr = (char*) AStr255;
		for (i = 0; i < length; i++,TheCharPtr++) {
			if (*TheCharPtr < ' ') {
				*TheCharPtr = '\0';
				return;
			}
		}
}

BoxcarInquire(BoxcarChannel)
/* Inquire what the voltage is on a boxcar ADC channel	*/
int				BoxcarChannel;
{
		char		AString[3];
		/* Note: only 3 characters are needed	*/

		sprintf(AString, "?%d\r" , BoxcarChannel);
		ibwrt (boxcarDVM, AString, 3L);
		if (ibsta & ERR) BoxcarError("Bad write in BoxcarInquire");
}

BoxcarInit()				/* call this only once, period.	*/
{

/*	Assign unique identifier to the device DVM and store in variable
	dvm.  Check for error.(ibfind error = negative value returned.)*/

	if ((boxcarDVM = ibfind ("boxcar")) < 0)
		BoxcarError("IBFIND error in BoxcarInit");

/*	Clear the device.  Check for error on each GPIB call to be safe.
	Note that GPIB status is available both through global variable
	ibsta and through the return values of all GPIB functions except
	IBFIND.*/

	if (ibclr (boxcarDVM) & ERR)
		BoxcarError("IBCLR error in BoxcarInit");

/*	Make CR-LF the EOS character*/

	ibeos (boxcarDVM, XEOS | REOS | '\n') ;

/* Set the boxcar into synchronous mode		*/

	ibwrt  (boxcarDVM,"MS\r",3L);
```

```
      if (ibsta & ERR) BoxcarError("IBWRT error in BoxcarInit");

      outBoxcarBuffIndex = 0; /* We have no characters in the
            (keyboard) buffer */
}

BoxcarError(ErrorString)
Str255          ErrorString;
{
            CtoPstr(ErrorString);
            ParamText(ErrorString,"\p","\p","\p");
            SysBeep(8);
            CautionAlert(GeneralAlert,NIL);
}


GetBoxcarVoltage2(BoxcarChannel1,BoxcarChannel2,
Voltage1Ptr,Voltage2Ptr)/* returns the voltages of two boxcar

            channels    */
int         BoxcarChannel1;
int         BoxcarChannel2;
float *Voltage1Ptr;
float *Voltage2Ptr;
{
      Str255          AStr255;
      float      atof();
      int             i;
      char       spr;

top:
      BoxcarInquire2(BoxcarChannel1,BoxcarChannel2);
      /* Inquire the voltages */

      for (i = 1; ; i++) {    /* Check if scan is done            */
            ibrsp(boxcarDVM,&spr);
            if (spr & ATN) break;
            if (i >= 50)      /* If no luck, resend the command   */
                  goto top;   /* But why does this happen so often?
      */
      }

      /* Double check: Did the "scan" only have one trigger,
            as it was supposed to?                    */
      ibwrt (boxcarDVM, "?N\r", 3L);
      if (ibsta & ERR) BoxcarError("IBWRT error in BoxcarInquire2 : 1");
      if (ibrd (boxcarDVM,AStr255,255L) & ERR)
      BoxcarError("IBRD error in GetBoxcarVoltage2 : 1");
      CleanUpString(AStr255);
      sscanf(AStr255,"%d",&i);
      if (i != 1) BoxcarError("Scan error in GetBoxcarVoltage2");
       /* N must = 1 */

      /* Decode the first voltage   */
      ibwrt (boxcarDVM, "N\r", 2L);
      if (ibsta & ERR) BoxcarError("IBRSP error in BoxcarInquire2 : 2");
      if (ibrd (boxcarDVM,AStr255,255L) & ERR)
      BoxcarError("IBRD error in GetBoxcarVoltage2 : 2");
```

```
	CleanUpString(AStr255);
	sscanf(AStr255,"%f",Voltage1Ptr);

	/* Decode the second voltage  */
	ibwrt (boxcarDVM, "N\r", 2L);
	if (ibsta & ERR) BoxcarError("IBRSP error in BoxcarInquire2 : 3");
	if (ibrd (boxcarDVM,AStr255,255L) & ERR)
	BoxcarError("IBRD error in GetBoxcarVoltage2 : 3");
	CleanUpString(AStr255);
	sscanf(AStr255,"%f",Voltage2Ptr);

	if (*Voltage1Ptr > 10.237 || *Voltage1Ptr < -10.237) {
		sprintf(JunkCharArray,
		"Got a bad voltage from the boxcar: %f volts",*Voltage1Ptr);
		BoxcarError(JunkCharArray);
		goto top;
	}
	if (*Voltage2Ptr > 10.237 || *Voltage2Ptr < -10.237) {
		sprintf(JunkCharArray,
		"Got a bad voltage from the boxcar: %f volts",*Voltage2Ptr);
		BoxcarError(JunkCharArray);
		goto top;
	}
}

BoxcarInquire2(BoxcarChannel1,BoxcarChannel2)
/* Scan the voltages on the boxcar ADC channels */
int		BoxcarChannel1;
int		BoxcarChannel2;
{
	char		AString[8];

	sprintf(AString, "SC%d,%d:1\r", BoxcarChannel1, BoxcarChannel2);
	ibwrt (boxcarDVM, AString, 8L);
	if (ibsta & ERR) BoxcarError("IBWRT error in BoxcarInquire2");
}

SendBoxChar(theChar)
char theChar;
/*
 Send characters obtained from the keyboard (NOT FROM DATA RUNS) to
 boxcar. This is done strangely, so listen up:  Rather then send a
 lot of single characters,we save up the characters in a buffer,
 outBoxcarBuff.  When theChar is a CR, then we stuff the whole buffer
 into the IEEE.  Big note: make sure outBoxcarBuffIndex is set
 equal to zero on opening the program. We also flush the buffer if
 it gets to OUTBOXBUFFERLENGTH - 1 characters, even if the user
 hasn't hit CR.  That's his tough luck.
*/

{
	if (theChar >= 32)
	{		/* Then theChar is just a normal character
	*/
		outBoxcarBuff[outBoxcarBuffIndex++] = theChar;
		if ( outBoxcarBuffIndex < OUTBOXBUFFERLENGTH - 1)
	return;
	}
```

```
        /* here if theChar is a CR (or a tab, etc.)
                                        OR if the buffer is full
       */

       outBoxcarBuff[outBoxcarBuffIndex++]  =  '\r';
             /* add EOS character*/
       /*DebugInt(outBoxcarBuffIndex);
       DebugString(outBoxcarBuff);*/
       ibwrt (boxcarDVM, outBoxcarBuff, (long) outBoxcarBuffIndex);
       if (ibsta & ERR) BoxcarError("IBWRT error in SendBoxChar");

       outBoxcarBuffIndex = 0;
}

EchoBoxcarOutput()
{
       extern           TEHandle         TEH;
       extern           char             dirty;
       int              spr;

       Str255           AStr255;

       ibrsp(boxcarDVM,&spr);
       if (spr & SRQI) {
             DebugStr("\pWay to go");
       } else {
             return;
       }

       /*if (ibrd (boxcarDVM,AStr255,255L) & ERR)
                         BoxcarError("IRD error in EchoBoxcarOutput");
       TEKey( theChar, TEH );          print it out
       ShowSelect();
       dirty = 1;*/
}
```

```
/*
      File Name:          nrutil_fixed.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            subroutine from Numerical Recipes
      Compiler:           Lightspeed C 3.01

      Utility:            This file contains routines used in the
                          Numerical Recipes programs

      Note:               Very few changes made; actually, only nrerror.
                          Routines for vector and free_vector added by
                          LWP.
                          Format slightly changed to match that of
arrays.c
                          in Trajectory View program, e.g., variables
                          renamed dm instead of m in the creation of a
                          double matrix.
                          Storage library (storage.h) now included by LWP
*/

#include   "stdio.h"
#include   "storage.h"

#define      GeneralAlert      255
extern       char  JunkCharArray[255];
void  free();

void  nrerror(error_text)
char  error_text[];
{
           sprintf(JunkCharArray,error_text,"Bad  initial  guesses");
           CtoPstr(JunkCharArray);
           ParamText(JunkCharArray,"\p","\p","\p");
           CautionAlert(GeneralAlert,0);

      return;
}



float  *vector(nl,nh)
int  nl,nh;
{
        float *v;

          v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
         if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int  *ivector(nl,nh)
int  nl,nh;
{
        int *v;

          v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
         if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}

double  *dvector(nl,nh)
int  nl,nh;
{
        double *v;

          v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
         if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}



float  **matrix(nrl,nrh,ncl,nch)
int  nrl,nrh,ncl,nch;
{
        int i;
        float **m;

          m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
           if (!m) nrerror("Allocation failure 1 in matrix()");
        m -= nrl;

         for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned)
               (nch-ncl+1)*sizeof(float));
                 if (!m[i]) nrerror("Allocation failure 2 in matrix()");
              m[i] -= ncl;
        }
        return m;
}

double  **dmatrix(nrl,nrh,ncl,nch)
int  nrl,nrh,ncl,nch;
{
        int i;
        double **m;

          m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
         if (!m) nrerror("allocation failure 1 in dmatrix()");
        m -= nrl;

         for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned)
               (nch-ncl+1)*sizeof(double));
                 if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
              m[i] -= ncl;
        }
        return m;
}

int  **imatrix(nrl,nrh,ncl,nch)
int  nrl,nrh,ncl,nch;
{
        int i,**m;
```

```
          m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
         if (!m) nrerror("allocation failure 1 in imatrix()");
        m -= nrl;

          for(i=nrl;i<=nrh;i++) {
                 m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
                 if (!m[i]) nrerror("allocation failure 2 in imatrix()");
               m[i] -= ncl;
        }
        return m;
}


float  **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float  **a;
int  oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
        int i,j;
        float **m;

          m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
         if (!m) nrerror("allocation failure in submatrix()");
        m -= newrl;

           for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

        return m;
}


void  free_vector(v,nl,nh)
float  *v;
int  nl,nh;
{
          free((char*) (v+nl));
}

void  free_ivector(v,nl,nh)
int  *v,nl,nh;
{
          free((char*) (v+nl));
}

void  free_dvector(v,nl,nh)
double  *v;
int  nl,nh;
{
          free((char*) (v+nl));
}


void  free_matrix(m,nrl,nrh,ncl,nch)
float  **m;
int  nrl,nrh,ncl,nch;
{
        int i;

          for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
         free((char*) (m+nrl));
}

void  free_dmatrix(m,nrl,nrh,ncl,nch)
double  **m;
int  nrl,nrh,ncl,nch;
{
        int i;

          for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
         free((char*) (m+nrl));
}

void  free_imatrix(m,nrl,nrh,ncl,nch)
int  **m;
int  nrl,nrh,ncl,nch;
{
        int i;

          for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
         free((char*) (m+nrl));
}



void  free_submatrix(b,nrl,nrh,ncl,nch)
float  **b;
int  nrl,nrh,ncl,nch;
{
         free((char*) (b+nrl));
}



float  **convert_matrix(a,nrl,nrh,ncl,nch)
float  *a;
int  nrl,nrh,ncl,nch;
{
         int i,j,nrow,ncol;
        float **m;

         nrow=nrh-nrl+1;
         ncol=nch-ncl+1;
          m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
          if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
           for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}



void  free_convert_matrix(b,nrl,nrh,ncl,nch)
float  **b;
int  nrl,nrh,ncl,nch;
```

```
{
          free((char*) (b+nrl));
}
```

```
/*
      File Name:          pleasewait.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            from LightSpeedC part of Miniedit program
      Compiler:           Lightspeed C 3.01

      Utility:            a utility function used by Miniedit
*/

#include  "Quickdraw.h"

extern  Cursor     waitCursor;

PleaseWait()
{
      SetCursor(  &waitCursor  );
}
```

```
/*
      File Name:          plot.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Mark J. Rosker
      Compiler:           Lightspeed C 3.01

      Utility:            this file takes care of ploting the data during
                          data acquisition

*/

#include "least_squares.h"

#include "Data Acquisition.h"

#define         DPSIZEINPIXELS        2
/* size of a data pixel--keep it an integer     */

#define         NOPOINTSOUTOFBOX            FALSE

extern      WindowPtr          PlotWindPtr;
extern      WindowRecord       PlotWindRec;


/*
BIG NOTE: All routines *** EXCEPT DrawPlotWind *** in this file
assume the current port is the plot window.
Forget this at your peril !
*/

typedef struct axes
{
      Rect                plotrect;
      int                          WidthPix;   /* width of plot rect, in
pixels     */
      int                          HeightPix;  /* height of plot rect, in
pixels     */
      float               xmin;         /* minimum of x axis, as a value
      */
      float               xmax;         /* maximum of x axis, as a value
      */
      float               ymin;         /* minimum of y axis, as a value
      */
      float               ymax;         /* maximum of y axis, as a value
      */
      float               xscale;       /* gives scale, in value per
pixels*/
      float               yscale;       /* gives scale, in value per
pixels*/
      int                          xdiv; /* number of "clean" divisions
      x*/
      int                          ydiv; /* number of "clean" divisions
      y*/
      int                          xdig; /* number of digits after decimal
x*/
      int                          ydig; /* number of digits after decimal
y*/
```

```
};

static       struct      axes  PlotAxes;
extern       Boolean                 DataTakenFlag;
extern       int                     NumOfDataPts;
extern       int                     RunInProgress;
extern       int                     ScanCounter;

extern       Str255                  JunkStr255;

extern       PicHandle         PlotWindPic;

OpenPlotWind()
{

      PlotWindPtr = GetNewWindow(PLOTWINDOWID, &PlotWindRec, -1L);
      (*(WindowPeek)PlotWindPtr).windowKind = PLOTWINDKIND;

      GetPlotAxesRect();
}

DrawPlotWind(DrawData,XData,YData,NumOfDataPts,NumYRows,CurrYRow,
      TheFunc, a, NumOfParams)
float             *XData;
float             **YData;
int                     NumYRows, CurrYRow;
int                     NumOfDataPts;
Boolean                 DrawData;
int                     NumOfParams;
double                  a[];
void              (*TheFunc) ();
{
      WindowPtr         TempPortPtr;

      GetPort(&TempPortPtr);              /*   Save the current port*/
      SetPort(PlotWindPtr);               /*   Point to plot window*/

      if (DataTakenFlag) EraseRect(&(*PlotWindPtr).portRect);

      if (PlotWindPic != NIL) KillPicture(PlotWindPic);
      PlotWindPic = OpenPicture(&(*PlotWindPtr).portRect);

      SetPlotAxesMinMax(XData,YData,NumOfDataPts,NumYRows,CurrYRow);

      /* sets all the scaling */


      if (DataTakenFlag && DrawData)
      {
      /*
      Only draw data points in if we've taken some, and this isn't
      a new run
      */
            ForeColor(redColor);

      DrawAllDataPoints(XData,YData,NumOfDataPts,NumYRows,CurrYRow);
      }
```

```
        ForeColor(greenColor);
        DrawAxes();
        DrawAxesLabels();
        ForeColor(blackColor);
        if (TheFunc != NIL)
        DrawAFunc(PlotAxes.xmin, PlotAxes.xmax, a, NumOfParams, TheFunc);


        ClosePicture();
        DrawPicture(PlotWindPic,&(*PlotWindPtr).portRect);
        SetWindowPic(PlotWindPtr, PlotWindPic);

        SetPort(TempPortPtr);           /*   Point to plot window   */
}

SetPlotAxesMinMax(XData,YData,NumOfDataPts,NumYRows,CurrYRow)
float *XData;
float **YData;
int         NumYRows, CurrYRow;
int         NumOfDataPts;
{
/*
   Set xmin, xmax, and xscale.  Same for y.
 */
        extern float XScale;
        int    i;
        double      DataXDiff;
        double      min;
        double      max;



   /*
Do x first. Same caluculation whether or not no data taken yet
We need at least one pixel per point
*/
        if (NumOfDataPts > DPSIZEINPIXELS * PlotAxes.WidthPix)
              DebugStr("\pPlot axis not big enough!");

        DataXDiff = (double) (XData[NumOfDataPts - 1] - XData[0]);

        min = (double) (XData[0]);
        max = (double) (XData[NumOfDataPts - 1]);

        scale(&min, &max, &(PlotAxes.xdiv), &(PlotAxes.xdig));
        /* does the scaling.  We don't use div,dig (yet)      */

        PlotAxes.xmin = min;
        PlotAxes.xmax = max;
        PlotAxes.xscale = (PlotAxes.xmax - PlotAxes.xmin)
        / PlotAxes.WidthPix;
        XScale = PlotAxes.xscale;

   /* now do y */
        if (DataTakenFlag)
        {           /* then calculate the real min, max */

              PlotAxes.ymin = YData[CurrYRow][0];
              PlotAxes.ymax = YData[CurrYRow][0];
```

```
          for (i = 1; i < NumOfDataPts; i++) {
               if (YData[CurrYRow][i] < PlotAxes.ymin)
                    PlotAxes.ymin = YData[CurrYRow][i];
               if (YData[CurrYRow][i] > PlotAxes.ymax)
                    PlotAxes.ymax = YData[CurrYRow][i];
          if(PlotAxes.ymin == 0.0 && PlotAxes.ymax == 0.0){
               PlotAxes.ymin = -10.0;
               PlotAxes.ymax =  10.0;
               }
          }

     }
     else
     { /* if no data yet, make the range +/- 10.0    */
          PlotAxes.ymin = -10.0;
          PlotAxes.ymax =  10.0;
     }

     min = (double) (PlotAxes.ymin);
     max = (double) (PlotAxes.ymax);

     scale(&min, &max, &(PlotAxes.ydiv), &(PlotAxes.ydig));
     /* does the scaling.  We don't use div,dig (yet)      */

     PlotAxes.ymin = min;
     PlotAxes.ymax = max;

     PlotAxes.yscale = (PlotAxes.ymax - PlotAxes.ymin)
     / PlotAxes.HeightPix;
}

DrawAFunc(xmin, xmax, a, NumOfParams, TheFunc)
double      xmin, xmax;
int         NumOfParams;
double      a[];
void  (*TheFunc) ();
{
     double      fitx,fity,junk;
     int         i;
     Point APoint;
     int         NumOfPts;
     double      dyda[MAXFITPARAMS];
     /* this is really just junk   */

     NumOfPts = 128;
     ForeColor(cyanColor);
     for (i = 0; i < NumOfPts; i++) {
          fitx = xmin + i * (xmax - xmin)/(NumOfPts - 1);
          (*TheFunc) (fitx,a,&fity,dyda,NumOfParams);
          GetPointFromData((float) fitx,(float) fity,&APoint);

          if (i == 0)
               MoveTo(APoint.h,APoint.v);
          else
               LineTo(APoint.h,APoint.v);
     }
}
```

```
DrawAxesLabels()                /* assumes scaling already done     */
{
      Str255            AStr255;
      int               i;
      float             x,y;
      Point        APoint;

   /* x labels     */
      for (i = 0; i <= PlotAxes.xdiv; i++) {

      /* Put tick marks in    */
            x   =   (PlotAxes.xmin * (PlotAxes.xdiv - i) / PlotAxes.xdiv)
                  + (PlotAxes.xmax * i / PlotAxes.xdiv);
            GetPointFromData(x,PlotAxes.ymin,&APoint);
            MoveTo(APoint.h,APoint.v);
            Line(0,5);

      /* Put labels in  */
            sprintf(AStr255,"%.*f",PlotAxes.xdig,x);
            CtoPstr(AStr255);
            Move(-StringWidth(AStr255)/2,15);
            DrawString(AStr255);
      }

   /* y labels     */
      for (i = 0; i <= PlotAxes.ydiv; i++) {
            y   =   (PlotAxes.ymin * (PlotAxes.ydiv - i) / PlotAxes.ydiv)
                  + (PlotAxes.ymax * i / PlotAxes.ydiv);
            GetPointFromData(PlotAxes.xmin,y,&APoint);
            MoveTo(APoint.h,APoint.v);
            Line(-5,0);

            sprintf(AStr255,"%.*f",PlotAxes.ydig,y);
            CtoPstr(AStr255);
            Move(-(StringWidth(AStr255)  +  3),5);
            DrawString(AStr255);
      }
}

DrawAxes()          /* doesn't draw labels  */
{
      Point APoint;
      /*FrameRect(&PlotAxes.plotrect);*/

      GetPointFromData(PlotAxes.xmin,PlotAxes.ymin,&APoint);
      MoveTo(APoint.h,APoint.v);
      GetPointFromData(PlotAxes.xmax,PlotAxes.ymin,&APoint);
      LineTo(APoint.h,APoint.v);
      GetPointFromData(PlotAxes.xmax,PlotAxes.ymax,&APoint);
      LineTo(APoint.h,APoint.v);
      GetPointFromData(PlotAxes.xmin,PlotAxes.ymax,&APoint);
      LineTo(APoint.h,APoint.v);
      GetPointFromData(PlotAxes.xmin,PlotAxes.ymin,&APoint);
      LineTo(APoint.h,APoint.v);

}
```

```
DrawAllDataPoints(XData,YData,NumOfDataPts,NumYRows,CurrYRow)
float *XData;
float **YData;
int         NumYRows,  CurrYRow;
int         NumOfDataPts;
{
      int               i;
      Point       APoint;
      Rect        ARect;

/*    for (i=0; i < NumOfDataPts; i++) {

            DrawADataPoint( (float) DP[i].DataX,  DP[i].DataY);

      }*/

      /* oldway to do it -- gives a line: */
      MoveTo(PlotAxes.plotrect.left,PlotAxes.plotrect.bottom);
      for (i = 0; i < NumOfDataPts; i++) {
            GetPointFromData((float)
XData[i],YData[CurrYRow][i],&APoint);
            if (i == 0) {
                  MoveTo(APoint.h,APoint.v);
            } else {
                  /*if  (PtInRect(APoint,&PlotAxes.plotrect))  */
                  LineTo(APoint.h,APoint.v);
            }
      }
}

DrawADataPoint(x,y)
float x,y;
{
      Point       APoint;
      Rect        ARect;

      GetPointFromData(x,y,&APoint);
      SetRect(&ARect,APoint.h  -  DPSIZEINPIXELS/2,APoint.v
      - DPSIZEINPIXELS/2,APoint.h  +  (DPSIZEINPIXELS  +  1)/2,
      APoint.v  +  (DPSIZEINPIXELS  +  1)/2);

      if  (NOPOINTSOUTOFBOX)  {
            if  (ARect.left     <  PlotAxes.plotrect.left)
                  ARect.left     =  PlotAxes.plotrect.left  +  1;
            if  (ARect.top      <=  PlotAxes.plotrect.top)
                  ARect.top      =  PlotAxes.plotrect.top  +  1;
            if  (ARect.right    >  PlotAxes.plotrect.right)
                  ARect.right    =  PlotAxes.plotrect.right  -  1;
            if  (ARect.bottom  >=  PlotAxes.plotrect.bottom  +  1)
                  ARect.bottom  =  PlotAxes.plotrect.bottom  +  1;
      }

      PaintRect(&ARect);

}

EraseADataPoint(x,y)
float x,y;
```

```
{
      Point       APoint;
      Rect        ARect;

      GetPointFromData(x,y,&APoint);
      SetRect(&ARect,APoint.h  -  DPSIZEINPIXELS/2,APoint.v
      - DPSIZEINPIXELS/2, APoint.h + (DPSIZEINPIXELS + 1)/2,
      APoint.v + (DPSIZEINPIXELS + 1)/2);
      if (NOPOINTSOUTOFBOX) {
            if (ARect.left    < PlotAxes.plotrect.left)
                  ARect.left    = PlotAxes.plotrect.left + 1;
            if (ARect.top     <= PlotAxes.plotrect.top)
                  ARect.top     = PlotAxes.plotrect.top + 1;
            if (ARect.right   > PlotAxes.plotrect.right)
                  ARect.right   = PlotAxes.plotrect.right - 1;
            if (ARect.bottom >= PlotAxes.plotrect.bottom + 1)
                  ARect.bottom  = PlotAxes.plotrect.bottom + 1;
      }

      EraseRect(&ARect);
}

GetPointFromData(x,y,APointPtr)
float       x,y;
Point *APointPtr;
/*
  Returns by APointPtr the correct Point for the input data pair.
 */
{
      int  dummy;
      float  fdummy;

/*    DebugFloat(x);
      fdummy = (x - PlotAxes.xmin) / PlotAxes.xscale;
      DebugFloat(fdummy);
      dummy = (int) fdummy;
      DebugInt(dummy);*/


      (*APointPtr).v  =  PlotAxes.plotrect.bottom
      - (int) ( (y - PlotAxes.ymin) / PlotAxes.yscale);
      (*APointPtr).h  =  PlotAxes.plotrect.left
      + (int) ( (x - PlotAxes.xmin) / PlotAxes.xscale);

}


GetPlotAxesRect()
 /*
Set up PlotAxes.plotrect, PlotAxes.WidthPix,
and PlotAxes.HeightPix
  */
{
      PlotAxes.plotrect.left  =  (*PlotWindPtr).portRect.left;
      PlotAxes.plotrect.top   =  (*PlotWindPtr).portRect.top;
      PlotAxes.plotrect.right =  (*PlotWindPtr).portRect.right;
      PlotAxes.plotrect.bottom     =  (*PlotWindPtr).portRect.bottom;
      InsetRect(&PlotAxes.plotrect,50,30);
```

```
        PlotAxes.WidthPix =  PlotAxes.plotrect.right
        -  PlotAxes.plotrect.left;
        PlotAxes.HeightPix       =  PlotAxes.plotrect.bottom
        -  PlotAxes.plotrect.top;
}
```

```
/*
      File Name:                        ReadSaveFile.c
      Version:                          2.0
      Project Name:                     DataAcquisition/Data  Acquire-047A
      Author:                                 L. W. Peng
      Date:                             June  15,1989
      Compiler:                         Lightspeed C   3.01

      Utility:                          Access/read/Create  data  files
without
                                        having to use the dialog box.
                                        Modified by LWP for use in 047 data
                                        acquisition  program

*/


*include  "stdio.h"
*include  "Data  Acquisition.h"

extern      float       *XData;
extern      float       **YData;
extern   char           volName[FILENAMELENGTH];
/*used to set HFS volume*/
extern   char          filename[FILENAMELENGTH];
/* file name of output  */
extern   char          filenameSafe[FILENAMELENGTH];
/* safe storage for output file name*/




int  SaveNewFile(filename,NumOfFilePts,CurrYRow)
/*Save data file-override file of same name*/
char  filename[FILENAMELENGTH];
int   NumOfFilePts;
int   CurrYRow;
{
      FILE  *filePtr;
      char  myString[255];
      int      k;

      filePtr = fopen (filename, "r+");
      fprintf(filePtr,"*  \r%s\t%s\n","Position","Signal");
      /*Header for compatibility with Write.c*/

      for(k=0;k<NumOfFilePts;k++)   {
                 if(fprintf (filePtr, "%f\t%f\n",XData[k],
                      YData[CurrYRow][k])==EOF)     {
                      printf ("End of File-jettisoning...");
                      exit (1);
                 }
      }
      fclose (filePtr);
}


int  CreateNewFile(filename)          /*create a totally new filename*/
char  filename[FILENAMELENGTH];
```

```
{
        FILE  *filePtr;
        char  myString[255];
        int      k;

        filePtr = fopen (filename, "w+");
        fclose  (filePtr);
}
```

```
/*
        File Name:          scale.c
        Version:            1.0
        Project Name:       DataAcquisition
        Authors:            Mark J. Rosker
        Compiler:           Lightspeed C 3.01

        Utility:            This file takes care of scaling the data so
                            that it always fills the screen
*/

scale(vmin, vmax, vdiv, vdig)
double      *vmin;
double      *vmax;
int         *vdiv;
int         *vdig;
{
        double      dv;
        int         i, j;

        i = 0;
        dv = *vmax - *vmin;
        if (dv == 0){
                *vmin = (double)((int)(*vmin - .5));
                *vmax = (double)((int)(*vmax + .51));
                *vdiv = 4;
                *vdig = 1;
                return(0);
        }

        if (dv != 0){
                while (dv <= 2){
                        j = 0;
                        i++;
                        dv = dv * 10;
                        *vmin = (*vmin)*10;
                        *vmax = (*vmax)*10;
                }
                while (dv > 20){
                        j = 1;
                        i++;
                        dv = dv/10;
                        *vmin = (*vmin)/10;
                        *vmax = (*vmax)/10;
                }
        }

/*
This is how Dan did it.  It seems that math lib has changed.
Now, (double) (int) x truncates abs(x).
This makes it more complicated.

        THIS IS THE OLD WAY
        if (*vmin != (double)((int)(*vmin))){
                *vmin = (double)((int)(*vmin - .49));
        }
        if (*vmax != (double)((int)(*vmax))){
                *vmax = (double)((int)(*vmax + .49));
        }

        New way to do this follows:
*/

        if (*vmin != (double) ((int) (*vmin)) ) {
              if (*vmin > 0.0 ) {
                    *vmin = (double) ((int) *vmin);
              } else if (*vmin < 0.0 ) {
                    *vmin = (double) ((int) (*vmin - 1.0));
              }
        }

        if (*vmax != (double) ((int) (*vmax)) ) {
              if (*vmax > 0.0 ) {
                    *vmax = (double) ((int) (*vmax + 1.0));
              } else if (*vmax < 0.0 ) {
                    *vmax = (double) ((int) *vmax);
              }
        }



        dv = *vmax - *vmin;
        switch  ((int)dv){
              case 26:    (*vmin)--;
              case 3 :    ;
              case  9:    ;
              case 27:    *vdiv = 3;
                                break;
              case 31:    (*vmin)--;
              case  4:    ;
              case  8:    ;
              case 16:    ;
              case 32:    *vdiv = 4;
                                break;
              case 19:    (*vmin)--;
              case  5:    ;
              case 10:    ;
              case 15:    ;
              case 20:    ;
              case 25:    *vdiv = 5;
                                break;
              case 22:    (*vmin)--;
              case 11:    ;
              case 17:    ;
              case 23:    ;
              case 29:    (*vmin)--;
              case  6:    ;
              case 12:    ;
              case 18:    ;
              case 24:    ;
              case 30:    *vdiv = 6;
                                break;
              case 13:    (*vmin)--;
              case  7:    ;
              case 14:    ;
              case 21:    ;
```

```
                case 28:      *vdiv = 7;
                                      break;
                default:      *vdiv = 4;
                                      break;
        }

        if (j){
                *vdig = 0;
                while (i > 0){
                        i--;
                        *vmin  =  (*vmin)*10;
                        *vmax  =  (*vmax)*10;
                }

        } else {
                *vdig = i;
                while (i > 0){
                        i--;
                        *vmin  =  (*vmin)/10;
                        *vmax  =  (*vmax)/10;
                }
        }
}
```

```
/*
	File Name:		Serial Comm.c
	Version:		1.0
	Project Name:		DataAcquisition
	Authors:		Mark J. Rosker
	Compiler:		Lightspeed C 3.01

	Utility:		This file takes care of serial communications
				which are used to control the actuators

	Notes:			Initially the boxcar was connected to the
				serial printer port, all sections related to
				the boxcar have been commented out since the
				boxcar is now interfaced with the IEEE488 (MD)
*/

#include "DeviceMgr.h"
#include "SerialDvr.h"
#include "DialogMgr.h"
#include "ControlMgr.h"
#include "TextEdit.h"
#include "Data Acquisition.h"

extern		char			actuatorInBuff[256];
extern		char			boxcarInBuff[256];
extern		DialogRecord		DlogRec;
extern		PortConfig	actuatorPort;
extern		PortConfig	boxcarPort;
extern		int			deviceSend;

InitSerialDrivers()
{
/*
   Call this routine only once, at the very begining of the program.
 */

	SerShk			*flags;
	int			dummy;

		/* Set up all the defaults settings:			*/
	actuatorPort.InRefNum		=	-6;
	/* actuator port input */
	actuatorPort.OutRefNum		=	-7;
	/* actuator port output */
	actuatorPort.baudSet		=	BAUDSET96;
	/* 19200 baud			*/
	actuatorPort.paritySet		=	PARITYSETN;
	/* No parity			*/
	actuatorPort.stopBitsSet	=	STOPBITSSET20;
	/* 2 stop bits			*/
	actuatorPort.dataBitsSet	=	DATABITSSET8;
	/* 8 data bits			*/
	actuatorPort.portSet		=	PORTSETMODEM;
	/* Use modem port	*/
	actuatorPort.whichDev		=	1;
	/* actuator 1			*/
	actuatorPort.echoOn			=	TRUE;
	/* Echo characters		*/
```

```
/*
      OLD SERIAL INTERFACE OF BOXCAR
      boxcarPort.InRefNum                          =     -8;
      boxcarPort.OutRefNum               =     -9;
      boxcarPort.baudSet                           =     BAUDSET192;
      boxcarPort.paritySet               =     PARITYSETN;
      boxcarPort.stopBitsSet             =     STOPBITSSET20;
      boxcarPort.dataBitsSet             =     DATABITSSET8;
      boxcarPort.portSet                           =     PORTSETPRINTER;
      boxcarPort.echoOn                  =     TRUE;
*/
      /* sPortA => modem port, sPortB => printer */
      if (RAMSDOpen(sPortA) != noErr) ErrorSComm(1);
      /*    if (RAMSDOpen(sPortB) != noErr) ErrorSComm(2);  */

      (*flags).fXOn  = 0;           /* Turn XOn/XOff on/off
            */
      (*flags).fCTS  = 1;           /* Turn hardware handshake on/off
      */
      (*flags).xOn   = 19;          /* XOn character = Cntrl-s
            */
      (*flags).xOff  = 17;          /* XOff character = Cntrl-q
            */
      (*flags).errs  = 0;           /* Errors which cause aborts
      */
      (*flags).evts  = 0;           /* Status changes that cause aborts
      */
      (*flags).fInX  = 0;           /* XOn/XOff input flow control flag
      */
      (*flags).fDTR  = 1;           /* Turn DTR handshake on/off
      */

      if (SerHShake(actuatorPort.OutRefNum, flags) != noErr)
        ErrorSComm(3);
      if (SerHShake(actuatorPort.InRefNum, flags) != noErr)
        ErrorSComm(3);
      if (SerSetBuf(actuatorPort.InRefNum, actuatorInBuff, 256)
        != noErr) ErrorSComm(4);

/*
      OLD HANDSHAKE FOR BOXCAR
      if (SerHShake(boxcarPort.OutRefNum, flags) != noErr)
        ErrorSComm(5);
      if (SerHShake(boxcarPort.InRefNum, flags) != noErr)
        ErrorSComm(5);
  NOTE: For some reason, the program frequently hangs up
   on the next line:
      if (SerSetBuf(boxcarPort.InRefNum, boxcarInBuff, 256) != noErr)
        ErrorSComm(6);
*/

      LoadCommOptions(&actuatorPort);
      /* loads all options for actuator port    */
/*
OLD BOXCAR
LoadCommOptions(&boxcarPort);
loads all options for boxcar port
```

```
*/
}

CloseSerialDrivers()
{
      /* sPortA => modem port, sPortB => printer */

      RAMSDClose(sPortA);
      RAMSDClose(sPortB);
}

int   DoCommunicationsMenu(item)
int         item;

{
      switch (item) {
      case  cmActuatorOptions:
      case  cmBoxcarOptions:
            DoOptionsDlog(item);
            break;
      case  cmSend:
            DoSendDlog();
            break;
      }
}

DoOptionsDlog(cmPort)
int         cmPort;
{
      DialogPtr   theDialog;
      int               itemHit;
      WindowPtr   tmpWindPtr;
      PortConfig  *myPort;

      int               baudSetTemp;
      int               paritySetTemp;
      int               stopBitsSetTemp;
      int               dataBitsSetTemp;
      int               portSetTemp;
      Boolean           echoOnTemp;

      GetPort(&tmpWindPtr);         /* Point at current window    */

      theDialog = GetNewDialog(OPTIONSDLOGID, &DlogRec,(WindowPtr)(-1));
      SetPort(theDialog);

      if (cmPort == cmActuatorOptions) {
            ParamText("\pActuator port configuration:","\P", "\P",
"\P");
            ShowIcon(theDialog, 27, 264);
            myPort = &actuatorPort;
      } else {
            ParamText("\pBoxcar port configuration:","\P", "\P", "\P");
            ShowIcon(theDialog, 27, 265);
            myPort = &boxcarPort;
      }

      baudSetTemp       =     (*myPort).baudSet;
```

```
        paritySetTemp       =      (*myPort).paritySet;
        stopBitsSetTemp     =      (*myPort).stopBitsSet;
        dataBitsSetTemp     =      (*myPort).dataBitsSet;
        portSetTemp         =      (*myPort).portSet;
        echoOnTemp          =      (*myPort).echoOn;

/*      Select controls where appropriate:               */
        ItemCheckMark(theDialog, baudSetTemp + BAUDOFFSET, 1);
        ItemCheckMark(theDialog, paritySetTemp + PARITYOFFSET, 1);
        ItemCheckMark(theDialog, stopBitsSetTemp + STOPBITSOFFSET, 1);
        ItemCheckMark(theDialog, dataBitsSetTemp + DATABITSOFFSET, 1);
        ItemCheckMark(theDialog, portSetTemp + PORTOFFSET, 1);
        ItemCheckMark(theDialog, 26, echoOnTemp); /* recall TRUE = 1*/

        ShowWindow(theDialog);
        frameOkbox(theDialog);

        do{
                ModalDialog(NIL, &itemHit);   /*calls filter before
handling*/
                switch (itemHit){
                        case 1 :                 /* Ok button
        */
                        case 2 :                 /* Cancel button        */
                                break;

                        case 3 :
                        case 4 :
                        case 5 :
                        case 6 :
                        case 7 :
                        case 8 :
                                ItemCheckMark(theDialog, baudSetTemp +
BAUDOFFSET, 0);
                                ItemCheckMark(theDialog, itemHit, 1);
                                baudSetTemp = itemHit - BAUDOFFSET;
                                break;

                        case 9 :
                        case 10 :
                        case 11 :
                                ItemCheckMark(theDialog, paritySetTemp
                                        + PARITYOFFSET,0);
                                ItemCheckMark(theDialog, itemHit, 1);
                                paritySetTemp = itemHit - PARITYOFFSET;
                                break;

                        case 12 :
                        case 13 :
                        case 14 :
                                ItemCheckMark(theDialog, stopBitsSetTemp
                                + STOPBITSOFFSET, 0);
                                ItemCheckMark(theDialog, itemHit, 1);
                                stopBitsSetTemp = itemHit - STOPBITSOFFSET;
                                break;

                        case 15 :
                        case 16 :
                                ItemCheckMark(theDialog,  dataBitsSetTemp
                                + DATABITSOFFSET, 0);
                                ItemCheckMark(theDialog, itemHit, 1);
                                dataBitsSetTemp = itemHit - DATABITSOFFSET;
                                break;

                        case 17 :
                        case 18 :
                                ItemCheckMark(theDialog,  portSetTemp
                                        + PORTOFFSET, 0);
                                ItemCheckMark(theDialog,  itemHit, 1);
                                portSetTemp = itemHit - PORTOFFSET;
                                break;

                        case 24 :/* What if you click on printer or modem
        */
                        case 25 :/* icon? Then set the correct port.
        */
                                ItemCheckMark(theDialog,  portSetTemp
                                        + PORTOFFSET, 0);
                                ItemCheckMark(theDialog,  itemHit - ICONOFFSET
                                        + PORTOFFSET, 1);
                                portSetTemp = itemHit - ICONOFFSET;
                                break;

                        case 26     :
                                echoOnTemp = !echoOnTemp;
                                ItemCheckMark(theDialog, itemHit, echoOnTemp);
                                break;


                }                               /*the case of what item was
hit*/
        }while (itemHit != 1 && itemHit != 2);

        CloseDialog(theDialog);
        SetPort(tmpWindPtr);

        if (itemHit != 2){              /* Unless it was cancel...
        */
                (*myPort).baudSet               =       baudSetTemp;
                (*myPort).paritySet             =       paritySetTemp;
                (*myPort).stopBitsSet   =       stopBitsSetTemp;
                (*myPort).dataBitsSet   =       dataBitsSetTemp;
                (*myPort).echoOn        =       echoOnTemp;

                if ((*myPort).portSet !=        portSetTemp)
                {               /* Must toggle both actuator and boxcar  */
                        /*CloseSerialDrivers();  */
                        if (actuatorPort.portSet ==  PORTSETMODEM) {
                                actuatorPort.portSet            =
                                        PORTSETPRINTER;
                                actuatorPort.InRefNum           =       -8;
                                actuatorPort.OutRefNum          =       -9;
                                boxcarPort.portSet                      =
        PORTSETMODEM;
                                boxcarPort.InRefNum                     =       -6;
                                boxcarPort.OutRefNum            =       -7;
                } else {
                        actuatorPort.portSet            =
        PORTSETMODEM;
                        actuatorPort.InRefNum           =       -6;
                        actuatorPort.OutRefNum          =       -7;
                        boxcarPort.portSet                      =
                                        PORTSETPRINTER;
                        boxcarPort.InRefNum                     =       -8;
                        boxcarPort.OutRefNum            =       -9;
                }
                (*myPort).portSet =     portSetTemp;
                LoadCommOptions(&actuatorPort);
                LoadCommOptions(&boxcarPort);

        } else {
                LoadCommOptions(myPort);
                /* In this case, only current one needs changing
        */
        }               /* end if portSet changed       */
        }               /* end if itemHit != 2          */

}

LoadCommOptions(myPort)
PortConfig  *myPort;

{
        int baud, parity, stopbits, databits;

        switch ((*myPort).baudSet)  {
                case BAUDSET3:
                        baud = baud300;
                        break;
                case BAUDSET12:
                        baud = baud1200;
                        break;
                case BAUDSET24:
                        baud = baud2400;
                        break;
                case BAUDSET96:
                        baud = baud9600;
                        break;
                case BAUDSET192:
                        baud = baud19200;
                        break;
                case BAUDSET576:
                        baud = baud57600;
                        break;
        }                               /* Switch to set baud rate    */

        switch ((*myPort).paritySet)  {
                case PARITYSETO:
                        parity = oddParity;
                        break;
                case PARITYSETE:
                        parity = evenParity;
                        break;
                case PARITYSETN:
```

```
                parity = noParity;
                break;
        }                               /* Switch to set parity */

        switch ((*myPort).stopBitsSet) {
                case STOPBITSSET10:
                        stopbits = stop10;
                        break;
                case STOPBITSSET15:
                        stopbits = stop15;
                        break;
                case STOPBITSSET20:
                        stopbits = stop20;
                        break;
        }                               /* Switch to set stop bits    */

        switch ((*myPort).dataBitsSet) {
                case DATABITSSET7:
                        databits = data7;
                        break;
                case DATABITSSET8:
                        databits = data8;
                        break;
        }                               /* Switch to set data bits    */


        SerReset((*myPort).InRefNum, baud + parity + stopbits + databits);
        SerReset((*myPort).OutRefNum, baud + parity + stopbits +databits);
}



DoSendDlog()
{
        DialogPtr       theDialog;
        int                     itemHit;
        WindowPtr       tmpWindPtr;

        int                     OnButton;

        GetPort(&tmpWindPtr);                   /* Point at current window
        */

        theDialog = GetNewDialog(SENDDLOGID, &DlogRec, (WindowPtr)(-1) );
        SetPort(theDialog);

/*      Select controls where appropriate:                      */

        if (deviceSend == DEVICESETACTUATOR) {
                OnButton = 3;
        } else {
                OnButton = 4;
        }

        ItemCheckMark(theDialog, OnButton, 1);
        ShowWindow(theDialog);
        frameOkbox(theDialog);
```

```
      do{
            ModalDialog(NIL, &itemHit);   /*calls filter before
handling*/
            switch (itemHit){
                  case 1 :                            /* Ok button
                  */
                  case 2 :                            /* Cancel button
            */
                        break;

                  case 3 :
                  case 4 :
                        ItemCheckMark(theDialog, OnButton, 0);
                        OnButton = itemHit;
                        ItemCheckMark(theDialog, OnButton, 1);
                        break;

                  case 6 :
                  case 7 :
                        ItemCheckMark(theDialog, OnButton, 0);
                        OnButton = itemHit - 3;
                        ItemCheckMark(theDialog, OnButton, 1);
                        break;


            }                                         /*the case of what
item was hit*/
      }while (itemHit != 1 && itemHit != 2);

      CloseDialog(theDialog);
      SetPort(tmpWindPtr);

      if (itemHit != 2){            /* Unless it was cancel...
      */
            switch (OnButton){
                  case 3 :
                        deviceSend = DEVICESETACTUATOR;
                        break;
                  case 4 :
                        deviceSend = DEVICESETBOXCAR;
                        break;
            }
      }
}

GetReturnedStr(ReturnedString, myPort, echoOn, WaitForReply)
/* returns a PASCAL string        */
unsigned char    ReturnedString[256];
PortConfig       *myPort;
Boolean                echoOn;
int                    WaitForReply;

{
      register    long i;                    /* index for chars */
      long        whatread;   /* how many chars are actually waiting */
      int               NumChars;
      char        c[256];
      extern            TEHandle        TEH;
```

```
        extern              char                dirty;

        unsigned long  GiveUpTime;          /* a time, in seconds
        */
        unsigned long  time();

        if  (NOCOMMUNICATE)  return;

        GiveUpTime = 4 + time(NULL);
        /* This is the time to give up (4 seconds from now)   */
        do {
             if  (SerGetBuf((*myPort).InRefNum,  &whatread)  !=  noErr)
                  DebugStr("\pDidn't  get  a  noErr");
                  /* anything there? */
             if  (WaitForReply  ==  DONTWAIT)
             /* then no need to loop forever          */
                  break;
             if  (time(NULL)  >  GiveUpTime)  {
             /* I assume that myPort is the actuator.  Boxcar now IEEE */
                  ParamText("\pThe  actuator  is  not  responding.
                  Is it turned on, and connected
properly?","\p","\p","\p");
                  NoteAlert(GeneralAlert,NIL);
             }

        } while (whatread == 0L) ;

        if (whatread > 0) {
             if (whatread > 255) whatread = 255L;
                  /*
                  if more than our buffer (shouldn't happen) coerce to
                  the buffer size, so we don't go destroying memory.
                  If there's more, it will have to wait until the next
                  pass
                  */


             if  (FSRead((*myPort).InRefNum,  &whatread,  c)  ==  noErr)
             {    /* no error, print the characters */
                  for  (i=0;  i<  whatread;  i++)
                  {
/*
We mask off the 8th bit, 'cause it doesn't really mean anything to us.
*/
                       c[i] = c[i] & 0x7f;
                       if  (echoOn)
                       {
                            TEKey( c[i], TEH );
                       }

                       if  (c[i]  <  32)
                       {              /* forces LF, etc. into EOS   */
                            ReturnedString[i]  =  '\0';
                            break;
                       }
                       else
                       {
                            ReturnedString[i]  =  c[i];
```

```
                              }
                    }

                    CtoPstr(ReturnedString);      /* return a PASCAL
string*/
                    if (echoOn)
                    {
                              ShowSelect();
                              dirty = 1;
                    }
          }
     }
}

ErrorSComm(ErrorCode)
int         ErrorCode;
{
     DebugStr("\pError in serial communications, with code:");
     DebugInt(ErrorCode);
}
```

```
/*
      File Name:          Utilities.c
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Mark J. Rosker
      Compiler:           Lightspeed C 3.01

      Utility:            This file contains small usefull routines
                          used throughout the program

      Note:               This file is not a subset of the one in LTPLOT
                          by the same name!
*/

#define MacTicker (*(long*)0x16A)

CountInHundreds(itop)
int   itop;
{
      long i;
      for (i=1; i <= (long) (100*itop) ; i++) {
            ;
      }
}

void sleep60(tics)
int tics;
{

register long beginticks,endticks;

      beginticks = MacTicker;
      endticks = beginticks + tics;
      while (MacTicker < endticks)
            ;
}

DebugDbl(value)
double value;
{
      char        out[255];
      sprintf(out,"The debug value is: %g",value);
      CtoPstr(out);
      DebugStr(out);
}

DebugFloat(value)
float value;
{
      char        out[255];
      sprintf(out,"The debug value is: %f",value);
      CtoPstr(out);
      DebugStr(out);
}

DebugChar(AChar)
char        AChar;
{
        char         out[255];
        sprintf(out,"The debug character is: %c",AChar);
        CtoPstr(out);
        DebugStr(out);
}

DebugString(AString)
char         AString[255];
{
        CtoPstr(AString);
        DebugStr(AString);
        PtoCstr(AString);
}

DebugInt(value)
int          value;
{
        char         out[255];
        sprintf(out,"The debug value is: %d",value);
        CtoPstr(out);
        DebugStr(out);
}

DebugLInt(value)
long         value;
{
        char         out[255];
        sprintf(out,"The debug value is: %ld",value);
        CtoPstr(out);
        DebugStr(out);
}


IsAnInt(theText)          /* return true is theText contains only
                             digits...Warning: does not look for sign */
char  theText[255];          /* assumed to be a Pascal type string */
{
        int    i;

        for (i = 1; i <= theText[0]; i++) {
                if ( (theText[i] < '0') || (theText[i] > '9' ) )
                        return(0);
        }
        return(1);
}

Capitalize(theCharPtr)
char  *theCharPtr;
{
        if ( (*theCharPtr >= 'a') && (*theCharPtr <= 'z') )
                *theCharPtr = *theCharPtr - ('a' - 'A');
}
```

```
/*
     File Name:          Write.c
     Version:            1.0
     Project Name:       DataAcquisition
     Authors:            Mark J. Rosker, some modifications by LWP
     Compiler:           Lightspeed C 3.01

     Utility:            This file writes the data into a file
*/

#include  "TextEdit.h"
#include  "StdFilePkg.h"
#include  "Data Acquisition.h"

extern      int         NumOfDataPts;
extern      TEHandle    TEH;
extern      char        dirty;


extern    char  filename[FILENAMELENGTH];
/*file name of output-made extern by LWP*/
int vRef;                 /* number of output volume    */

WriteData(XData,YData,NumOfDataPts,NumYRows,CurrYRow)
float *XData;
float **YData;
int          NumYRows,  CurrYRow;
int          NumOfDataPts;

/*
 Write the data out to the given filename
 */

{
     int    refNum;
     int    DPCounter,  i;
     int          WriteErrors;
     OSErr        FError;
     Point        where;
     SFReply  reply;
     char   outstr[80];
     char   volName[FILENAMELENGTH];
     long   count,  numbyte,  filePos;
     Ptr    ptrBuff;

     where.v  =  300;
     where.h  =  180;
     WriteErrors  =  0;         /* assume no write error      */

     MoveTo(5,5);

     MakeDefaultFilename();

     CtoPstr(filename);
     SFPutFile(where,"\PSave  current  data  set  as:",filename,NIL,
          &reply);
     if  (reply.good)  {
          for  (i  =  0;  i  <  FILENAMELENGTH;  i++)
```

```
                filename[i] = reply.fName[i];
            vRef = reply.vRefNum;

            FSDelete(filename,vRef);
            count = 2000L ;
            Create  (filename,vRef,'CGRF','CGTX');
            FSOpen  (filename,vRef,&refNum);
            Allocate  (refNum, &count);
            SetFPos(refNum, 1, 0L);

            sprintf(outstr,"*\rPosition\tSignal\r");
            numbyte = (long) strlen(outstr);
            ptrBuff = (Ptr) &outstr;
            FSWrite(refNum,&numbyte,ptrBuff);
            if (numbyte != (long) strlen(outstr)) WriteErrors++;
            for (DPCounter = 0; DPCounter < NumOfDataPts; DPCounter++)
            {     /* Output value in volts      */
                  sprintf(outstr,"%.0f\t%.4f\r",XData[DPCounter],
                        YData[CurrYRow][DPCounter]);
                  numbyte = (long) strlen(outstr);
                  ptrBuff = (Ptr) &outstr;
                  FSWrite(refNum,&numbyte,ptrBuff);
                  if (numbyte != (long) strlen(outstr)) WriteErrors++;
            }
            FSClose(refNum);
            FlushVol(volName, vRef);

            if (WriteErrors) {
                  sprintf(outstr,"Warning: %d write
errors\n",WriteErrors);
                  TEInsert( outstr, strlen(outstr), TEH );
                  ShowSelect();
                  dirty = 1;
                  SysBeep(3);
            }
      }
}

MakeDefaultFilename()
{
      /*tm              *localtime();*/
      char              *ctime();
      unsigned long     time();
      unsigned long     theTime;
      int                     i;
      char              tempname[FILENAMELENGTH];
      extern      int         DataRunParameterType;
      extern      Str255      ParamFileName[5];


      time(&theTime);
      theTime = theTime - (60 * 60 * 5);
/*
      For some incredible reason, we have to subtract 5 hours ???
      off current time for ctime to work properly.  Bizarre.
*/

      strcpy(tempname,ctime(&theTime));
```

```
        for (i =  0; i <= 1; i++) {
                filename[i] = tempname[8+i]; /* copy date of month   */
                if (filename[i] == ' ') filename[i] = '0';
        }
        for (i = 2; i <= 4; i++) {
                filename[i] = tempname[i + 2];      /* copy month of year
        */
        }
        for (i = 5; i <= 6; i++) {
                filename[i] = tempname[i + 17];     /* copy year
                */
        }
        for (i = 8; i <= 15; i++) {
                filename[i] = tempname[i + 3];      /* copy year
                */
                if (filename[i] == ':') filename[i] = ';';
        }

        filename[16] = '\0';
        filename[7]  = ' ';

        switch (DataRunParameterType)  {
                case dcAmpAC:
                        strcat(filename," Amp AC");
                        break;
                case dcCPMAC:
                        strcat(filename," CPM AC");
                        break;
                case dcICN:
                        strcat(filename," ICN");
                        break;
                case dcMixedSpectrum:
                        strcat(filename," Mix Sp");
                        break;
                case dcSHGSpectrum:
                        strcat(filename," SHG Sp");
                        break;
                case dcNaI:
                        strcat(filename," NaI");
                        break;
                case dcXCorr:
                        strcat(filename," XCorr");
                        break;
        }
}
```

```
/*
     File Name:          Data Acquisition.h
     Version:            1.0
     Project Name:       DataAcquisition
     Authors:            Mark J. Rosker, based on LightSpeedC Miniedit
                         project
     Compiler:           Lightspeed C 3.01

     Utility:            This header file contains definitions for most
                         variables and constants of the Data Acquisition
                         program
*/


#define     COMMWINDKIND             8
/* Code for types of windows used.  This is RS232 comm      */
#define     EDITWINDKIND             9
/* And this is a textedit window                                    */
#define     PLOTWINDKIND             10
/* and this is a plot window.  All added by mjr             */


#define  COMMWINDOWID          128          /* named changed mjr*/
#define     PLOTWINDOWID             129


#define     FitAlert           254                 /* added mjr     */
#define     GeneralAlert             255              /* added mjr
     */
#define  ErrorAlert                  258
#define     AdviseAlert        257

/* resource IDs of menus */
#define  appleID               128
#define  fileID                129
#define  editID                130
#define  communicateID         131
#define     datacollectionID   132
#define     fitID              133

/* Edit menu command indices */
#define  undoCommand           1
#define  cutCommand                  3
#define  copyCommand                 4
#define  pasteCommand          5
#define  clearCommand          6

/* Menu indices */
#define  appleM                0
#define  fileM                 1
#define  editM                 2
#define  communicateM          3
#define  datacollectionM       4
#define  fitM                        5

#define     NUMOFMENUS         6
/* added mjr; set equal to number of windows     */
```

```
*define  fmNew              1
*define  fmOpen             2
*define  fmClose            4
*define  fmSave             5
*define  fmSaveAs           6
*define  fmRevert           7
*define  fmPageSetUp                    9
*define  fmPrint            10
*define  fmQuit             12

*define  aaSave             1
*define  aaDiscard          2
*define  aaCancel           3

*define  cmActuatorOptions              1
*define  cmBoxcarOptions    2
*define  cmSend             3

*define  dcParameters       1
*define  dcStart            2
*define  dcPause            3
*define  dcAmpAC            5
*define  dcCPMAC            6
*define  dcICN              7
*define  dcMixedSpectrum    8
*define  dcSHGSpectrum      9
*define dcNal                           10
*define dcXCorr                         11
*define  dcSaveParams       13
*define  dcSaveCricketGraph 15

*define     ftGaussian      1
*define     ftLorentzian                2
*define     ftHypSecant     3
*define     ftSinusoidal                4
*define     ftExpStep       6
*define     ftAZmodel       7
*define     ftMJRmodel      8
*define     ftTest          9
*define ftBiExp             10
*define  ftSingExp          11
*define ftLine              12

*define  SBarWidth          15

*ifndef  NULL
*define  NULL  0L
*endif

/*   all of the following was added by MJR     */

*define     BAUDSET3                    1
*define     BAUDSET12                   2
*define     BAUDSET24                   3
*define     BAUDSET96                   4
*define     BAUDSET192                  5
*define     BAUDSET576                  6
```

```
*define     PARITYSETO              1
*define     PARITYSETE              2
*define     PARITYSETN              3

*define     STOPBITSSET10           1
*define     STOPBITSSET15           2
*define     STOPBITSSET20           3

*define     DATABITSSET7                    1
*define     DATABITSSET8                    2

*define     PORTSETMODEM                    1
*define     PORTSETPRINTER          2

*define     DEVICESETACTUATOR       1
*define     DEVICESETBOXCAR         2

*define     BAUDOFFSET              2
/* Add BAUDOFFSET to each BAUDSET (e.g.,  */
*define     PARITYOFFSET            8
/* BAUDSET96) to gets its DITL item number      */
*define     STOPBITSOFFSET      11 /* Similarly for the others.*/
*define     DATABITSOFFSET      14
*define     PORTOFFSET          16
*define     ICONOFFSET          23

*define  OPTIONSDLOGID          29859
*define  SENDDLOGID                     11448
*define  PARAMETERSDLOGID       20486
*define  FITDLOGID              12322


typedef struct {
      int  InRefNum;
      /* Reference number of current input driver
      (-6 = modem, -8 = printer)    */
      int  OutRefNum;
      /* Reference number of current output driver
      (-7 = modem, -9 = printer)    */
      int   baudSet;
      /* Baud rate of current port, as defined in Serial Comm.c*/
      int   paritySet;
      /* Parity setting of current port, as defined in Serial Comm.c*/
      int   stopBitsSet;
      /* Number of stop bits of current port, as defined in Serial
      Comm.c          */
      int   dataBitsSet;
      /* Number of data bits of current port, as defined in Serial
      Comm.c          */
      int   portSet;
      /* Device setting, PORTSETMODEM or PORTSETPRINTER          */
      int   whichDev;          /* e.g., 1 for actuator *1, etc.   */
      Boolean echoOn;          /* TRUE if characters should be echoed
      before transmission out port */
} PortConfig;


/* These are the three allowed states of RunInProgress:     */
```

```
*define          NOTINPROGRESS     0
*define          INPROGRESS        1
*define          INPAUSE                  2


*define     NIL   0L
*define     NOCOMMUNICATE     FALSE
/* A debugging tool; put TRUE so no RS232/IEEE commun.      */

*define     DONTWAIT              0
/* values used in data collection loops; should I wait      */
*define     WAIT                  1
/* for a response from box (loop forever) or give up? */

/* code for CheckOnActPos parameter.  See collection.c
     Note:      Actuator position is always checked before
     the first data point, regardless.*/
*define     CHECKONLYATZERO       0
/* Never verify the actuator is where you think it is */
*define     CHECKNOLOOP           1
/* Check right after a move. Puts up a dialog if too big    */
*define     CHECKLOOP             2
/* Check act. position immed. after taking boxcar data; loop*/

*define MAXACTERROR                     5
/* Largest actuator error in counts allowed before          */
                            /* taking a pt              */

*define     FILENAMELENGTH        64    /*added LWP */
```

```
/*
	File Name:		DrInterface.h
	Version:		1.0
	Project Name:		DataAcquisition
	Authors:		Provided by National Instruments
	Compiler:		Lightspeed C 3.01

	Utility:		This header file contains definitions for most
				variables and constants of the Lab Driver
				IEEE488 interface card

	Notes:			This file is the include file for the interface
				between the *LI and the Driver.
				It MUST be included in both the ends of the
				interface
*/


typedef struct StatusBlk{
	int	ibsta;
	int	iberr;
	int	ibret;					/* The four GPIB */
	long	ibcnt;					/* status variables */
} StatusBlk;

typedef struct gpibBlock	{
	StatusBlk *statusBlk;
	int		id;
	int		controlVar;	/* Control variable for some	*/
					/* functions, to indicate nature */
					/* of action			*/
	Ptr		IOBufPtr;	/* Pointer to the buffer in	*/
					/* user area, during I/O calls
	*/
	long	IOCount;		/* I/O byte count
	*/
} gpibBlock;

enum {
	ibCAC, ibCMD, ibGTS, ibIST, ibLOC,
	ibONL, ibRD, ibRPP, ibRSC, ibPPC, ibWAIT,
	ibRSV, ibSIC, ibSRE, ibWRT, ibLLO,
	ibDMA, ibEOT, ibPAD, ibSAD, ibEOS,
	ibTRG, ibCLR, ibPCT, ibRSP, ibFIND,
	ibTMO, ibPOKE, ibSTAT};

extern int ibsta;
extern int iberr;
extern long ibcnt;
```

```
/*
      File Name:          GPIBres.h
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Provided by National Instruments
      Compiler:           Lightspeed C 3.01

      Utility:            This header file contains information
                          for the resources used for the IEEE488
                          interface card

      Notes:              The resources should be opened with
                          GetNamedResource since the resources
                          may be moved in to the system file, in
                          which case the ID's may be changed
                          by the installer. Use the following names.
*/

#define  DRIVER_DNAME          "\p.GPIB Driver"/* resource name    */

#define  BRD_NAMES_NAM    "\pGPIB-BRDnames"
#define  DEV_NAMES_NAM    "\pGPIB-DEVnames"
#define  BUS_NAMES_NAM    "\pGPIB-BUSnames"       /* for IBCONF */

#define  BRD_DATA_NAM     "\pGPIB-BRDdata"
#define  DEV_DATA_NAM     "\pGPIB-DEVdata"


typedef struct {
      int         Rev;
      uint8 Cnt;
      uint8 Size;
      struct board BoardData[NBOARDS];
      } BoardResource;

typedef struct {
      int         Rev;
      uint8 Cnt;
      uint8 Size;
      struct device DeviceData[NDEVICES];
      } DeviceResource;
```

```
/*
      File Name:          IEEE.h
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Provided by National Instruments
      Compiler:           Lightspeed C 3.01

      Utility:            This header file contains definitions for most
                          variables and constants of the IEEE488 interface
*/
/* C Declarations                                                 */
/*                                                                */
/* You MUST include the following three declarations in          */
/* your program.                                                  */
/*                                                                */
/* Status variables declared public by *cib*.obj.                */

extern  int     ibsta;  /* status word                            */
extern  int     iberr;  /* GPIB error code                        */
extern  long    ibcnt;  /* number of bytes sent                   */

/* Optionally include the following declarations in your         */
/* program.                                                       */

/* GPIB Commands                                                  */

*define UNL  0x3f  /* GPIB unlisten command                      */
*define UNT  0x5f  /* GPIB untalk command                        */
*define GTL  0x01  /* GPIB go to local                           */
*define SDC  0x04  /* GPIB selected device clear                 */
*define PPC  0x05  /* GPIB parallel poll configure               */
*define GET  0x08  /* GPIB group execute trigger                 */
*define TCT  0x09  /* GPIB take control                          */
*define LLO  0x11  /* GPIB local lock out                        */
*define DCL  0x14  /* GPIB device clear                          */
*define PPU  0x15  /* GPIB parallel poll unconfigure             */
*define SPE  0x18  /* GPIB serial poll enable                    */
*define SPD  0x19  /* GPIB serial poll disable                   */
*define PPE  0x60  /* GPIB parallel poll enable                  */
*define PPD  0x70  /* GPIB parallel poll disable                 */

/* GPIB status bit vector :                                       */
/*        global variable ibsta and wait mask                     */

*define ERR     (1<<15) /* Error detected                         */
*define TIMO    (1<<14) /* Timeout                                */
*define END     (1<<13) /* EOI or EOS detected                    */
*define SRQI    (1<<12) /* SRQ detected by CIC                    */
*define RQS     (1<<11) /* Device needs service                   */
*define CMPL    (1<<8)  /* I/O completed                          */
*define LOK     (1<<7)  /* Local lockout state                    */
*define REM     (1<<6)  /* Remote state                           */
*define CIC     (1<<5)  /* Controller-In-Charge                   */
*define ATN     (1<<4)  /* Attention asserted                     */
*define TACS    (1<<3)  /* Talker active                          */
*define LACS    (1<<2)  /* Listener active                        */
*define DTAS    (1<<1)  /* Device trigger state                   */
*define DCAS    (1<<0)  /* Device clear state                     */
```

```
/* Error messages returned in global variable iberr            */

*define EDVR  0  /* DOS error                                  */
*define ECIC  1  /* Function requires GPIB board to be CIC     */
*define ENOL  2  /* Write function detected no Listeners       */
*define EADR  3  /* Interface board not addressed correctly*/
*define EARG  4  /* Invalid argument to function call          */
*define ESAC  5  /* Function requires GPIB board to be SAC     */
*define EABO  6  /* I/O operation aborted                      */
*define ENEB  7  /* Non-existent interface board               */
*define EOIP 10  /* I/O operation started before previous      */
                 /* operation completed                    */
*define ECAP 11  /* No capability for intended operation       */
*define EFSO 12  /* File system operation error                */
*define EBUS 14  /* Command error during device call           */
*define ESTB 15  /* Serial poll status byte lost               */
*define ESRQ 16  /* SRQ remains asserted                       */

/* EOS mode bits                                                */

*define BIN  (1<<12) /* Eight bit compare                       */
*define XEOS (1<<11) /* Send EOI with EOS byte                  */
*define REOS (1<<10) /* Terminate read on EOS                   */

/* Timeout values and meanings                                  */

*define TNONE    0    /* Infinite timeout (disabled)            */
*define T10us    1    /* Timeout of 10 us (ideal)               */
*define T30us    2    /* Timeout of 30 us (ideal)               */
*define T100us   3    /* Timeout of 100 us (ideal)              */
*define T300us   4    /* Timeout of 300 us (ideal)              */
*define T1ms     5    /* Timeout of 1 ms (ideal)                */
*define T3ms     6    /* Timeout of 3 ms (ideal)                */
*define T10ms    7    /* Timeout of 10 ms (ideal)               */
*define T30ms    8    /* Timeout of 30 ms (ideal)               */
*define T100ms   9    /* Timeout of 100 ms (ideal)              */
*define T300ms  10    /* Timeout of 300 ms (ideal)              */
*define T1s     11    /* Timeout of 1 s (ideal)                 */
*define T3s     12    /* Timeout of 3 s (ideal)                 */
*define T10s    13    /* Timeout of 10 s (ideal)                */
*define T30s    14    /* Timeout of 30 s (ideal)                */
*define T100s   15    /* Timeout of 100 s (ideal)               */
*define T300s   16    /* Timeout of 300 s (ideal)               */
*define T1000s  17    /* Timeout of 1000 s (maximum)            */

/* Miscellaneous                                                */

*define S     0x08    /* parallel poll sense bit                */
*define LF    0x0a    /* ASCII linefeed character               */
```

```
/*
      File Name:          least_squares.h
      Version:            1.0
      Project Name:       DataAcquisition
      Authors:            Mark J. Rosker
      Compiler:           Lightspeed C 3.01

      Utility:            This header file defines the maximum
                          numbers of parameters to be used by
                          the NLS fitting routine
*/

*define MAXFITPARAMS 20 /* Set to largest number of fit parameters */
```

```
/*
      File Name:         NRUTIL.h
      Version:           1.0
      Project Name:      DataAcquisition
      Authors:           Numerical Recipes
      Compiler:          Lightspeed C 3.01

      Utility:           This header file contains declarations
                         of routines used in Numerical Recipes
*/

float  *vector();
float  **matrix();
float  **convert_matrix();
double  *dvector();
double  **dmatrix();
int  *ivector();
int  **imatrix();
float  **submatrix();
void  free_vector();
void  free_dvector();
void  free_ivector();
void  free_matrix();
void  free_dmatrix();
void  free_imatrix();
void  free_submatrix();
void  free_convert_matrix();
void  nrerror();
```

```
/*
      File Name:         sys.h
      Version:           1.0
      Project Name:      DataAcquisition
      Authors:           Provided by National Instruments
      Compiler:          Lightspeed C 3.01

      Utility:           This header file contains definitions for most
                         variables and constants of needed for the
                         driver of the IEEE488 board
*/

/*
  @(*) sys.h      - system definitions          (:ts=8)
 */

*include  <OSUtil.h>
*include  <timeMgr.h>
*include  <unix.h>

*ifndef  NULL
*define  NULL  0L
*endif

*define  SystemIsSE  3
*define  SystemIsNB  4

*define  Driver
*define  INTR

*ifdef  Driver
*define  pprintf
*endif

extern   int       ibsta;  /* status word                          */
extern   int       iberr;  /* GPIB error code                      */
extern   long      ibcnt;  /* number of bytes sent                 */

*define  NBOARDS          6
*define  NDEVICES         16
*define  MAX_NAME_LENGTH 16          /* including \0 terminator */

typedef      unsigned char      uint8;
typedef            unsigned short      uint16;
typedef      unsigned long      uint32;
typedef      uint8              *faddr_t;   /* far character pointer*/
typedef      uint16                   *fshort_t;  /* far uint16 pointer
      */

typedef union { struct { uint8 imrReg1, imrReg2; } imr;
             int imrs;
      }     imask_t;

/* write only hardware registers      */
struct  woregs  {
      char     w_admr;      /*      address mode register         */
      char     w_auxra;     /*      auxiliary register A          */
      char     w_imr1;       /*      interrupt mask register 1    */
```

```
        char     w_imr2;      /*        interrupt mask register 2    */
        char     w_imr3;   /*    configuration register         */
};


/* read only hardware registers      */
struct roregs {
        char     r_isr1;      /*        interrupt status reg 1       */
        char     r_isr2;      /*        interrupt status reg 2       */
        char     r_adsr;      /*        address status register      */
};



/*
Board Structure: b_uflags must be the first field in the internal
section and b_sptimo must be the first field in the user
section
*/

struct      board {

/* user secton */

        uint16      b_uflags;   /* user flags                     */
        uint8 b_pad;            /* primary address of board   */
        uint8 b_sad;            /* secondary address of board       */
        uint8 b_eos;            /* end of string byte         */
        uint8       b_tmo;      /* timeout value              */
        uint8 b_ppe;            /* parallel poll enable byte  */
        uint8 b_rsvcic;         /* attempt to take control    */


        /* internal portion */
        uint8             b_sptimo;/* timeout value for fast commands */
        uint8             spe_buf[6]; /* serial poll enable buffer */
        uint8             spe_cnt;    /* size of spe_buf */
        uint8             sp_disable[3];/* serial poll disable buffer */
        uint8                   auxmra;     /* software copy of auxmra */
        uint8             b_id;       /* id of the board */
        uint16                  b_state;    /* internal board state */
        uint16                  b_iberr;
        uint32                  b_ibcnt;
        TMTask                  tmTask;     /* time manager structure */
        imask_t                 b_wmask;
        struct                  roregs b_ro;      /* read only register */
        struct                  woregs b_wo;
        union {           fshort_t  wb_piobuf;/* far address for PIO */
                                faddr_t   cb_piobuf;     /* far address for
PIO */
                  } io_addr;
        uint16      b_piotcnt;  /* total count    */
        uint16      b_piocnt;         /* count so far              */
        uint8 b_pioop;          /* op type: P_READ or P_WRITE
        */
        uint8 last_byte;        /* save the last byte for word
        */
```

```
                                /* operations and odd request
      */
      char  *lastaddr;          /* address used for to check if
      */
                                /* repeat addressing is needed
      */
      uint16      Pio;          /* I/O mode
      */
      uint16   timedout;        /* address of the board timeout flag
      */
};


#define SPQSZ 5                         /* max serial poll queue size
      */

/* device structure */
struct      device      {
      uint16      d_uflags;             /* user defined status flags  */
      uint8 d_pad;                      /* primary address
      */
      uint8 d_sad;                      /* secondary address          */
      uint8 d_eos;                      /* end of string character    */
      uint8 d_bna;                      /* access board               */
      uint16      d_tmo;                /* timeout value              */

      /* internal device fields not configurable by the user          */
      uint16      d_state;              /* internal state of device   */
      char  d_ladstr[6];        /* listen address string              */
      char  d_tadstr[6];        /* talk address string                */
      uint8 d_adsz;                     /* length of address string   */
      uint8 d_spsz;                     /* length of serial poll string
      */
      uint8 d_spq[SPQSZ];       /* serial poll queue                  */
      uint8 d_spqcnt;           /* number of entries in spq           */
      uint8 d_rqsmask;          /* mask for rqs wakeup                */
      uint8 d_rqsval;           /* value for rqs wakeup
      */
};
extern struct device bdevice[];


/* The user status flags (returned from bstat) */

#define  U_ERR        0x8000    /* error occurred */
#define  U_TIMO       0x4000    /* timeout */
#define  U_END        0x2000    /* EOI detected */
#define  U_SRQI       0x1000    /* SRQ interrupt */
#define  U_RQS        0x800     /* device is requesting service */
#define  U_CMPL       0x100     /* dma completed */
#define  U_ADSC       0x400     /* address status change */
#define  U_LOK        0x80      /* local lockout state */
#define  U_REM        0x40      /* remote state */
#define  U_CIC        0x20      /* controller in charge */
#define  U_ATN        0x10      /* ATN line */
#define  U_TACS       0x8       /* talker active */
#define  U_LACS       0x4       /* listener active */
#define  U_DTAS       0x2       /* device trigger state */
```

```
*define  U_DCAS       0x1        /* device clear state */

*define  ERR    U_ERR      /*  error occurred */
*define  TIMO   U_TIMO     /*  timeout */
*define  END    U_END      /*  EOI detected */
*define  SRQI   U_SRQI     /*  SRQ interrupt */
*define  RQS    U_RQS      /*  device is requesting service */
*define  CMPL   U_CMPL     /*  dma completed */
*define  ADSC   U_ADSC     /*  address status change */
*define  LOK    U_LOK      /*  local lockout state */
*define  REM    U_REM      /*  remote state */
*define  CIC    U_CIC      /*  controller in charge */
*define  ATN    U_ATN      /*  ATN line */
*define  TACS   U_TACS     /*  talker active */
*define  LACS   U_LACS     /*  listener active */
*define  DTAS   U_DTAS     /*  device trigger state */
*define  DCAS   U_DCAS     /*  device clear state */

/* Error Codes    */

*define      EDVR     0     /* operating system error */
*define  ECIC         1     /* not controller in charge */
*define  ENOL         2     /* write error (no listeners) */
*define  EADR         3     /* CIC and not addressed before I/O */
*define  EARG         4     /* bad argument to function call */
*define  ESAC         5     /* not SAC */
*define  EABO         6     /* operation aborted (timeout) */
*define  ENEB         19 /* Non existant board */
*define  EDMA         8     /* DMA hardware error detected */
*define  ECAP         11 /* no capability for intended operation*/
*define EFSO        12 /* file system error */
*define  EBUS         14 /* GPIB bus error */
*define  ESTB         15 /* serial poll bytes queue overflow */
*define  ESRQ         16 /* SRQ stuck on */
*define  EASC         17 /* address status change during I/O command */
*define  EDC          18 /* DCAS occurred during I/O command */

/* timeout values */
*define  TNONE     0               /* timeout disabled */
*define  T10us     1               /* timeout of 10 us */
*define  T30us     2               /* timeout of 30 us */
*define  T100us    3               /* timeout of 100 us */
*define  T300us    4               /* timeout of 300 us */
*define  T1ms            5               /* timeout of 1 ms */
*define  T3ms            6               /* timeout of 3 ms */
*define  T10ms     7               /* timeout of 10  ms */
*define  T30ms     8               /* timeout of 30 ms */
*define  T100ms    9               /* timeout of 100 ms */
*define  T300ms    10              /* timeout of 300 ms */
*define  T1s       11              /* timeout of 1 s */
*define  T3s       12              /* timeout of 3 s */
*define  T10s            13              /* timeout of 10 s */
*define  T30s            14              /* timeout of 30 s */
*define  T100s     15              /* timeout of 100 s */
*define  T300s     16              /* timeout of 300 s */
*define  T1000s    17              /* timeout of 1000 s */
```

Appendix 3

The configuration of the NI488 board

The configuration procedure of the board is the following:
a) Run IBCONF program supplied by National Instruments.
b) Under the devices menu choose dev7 and select Change Name to "boxcar."
c) Under the devices menu choose dev13 and select Change Name to "oma."
d) Under the buses menu select bus4 and click on boxcar and oma.
e) Quit IBCONF and save the changes made.
f) Restart the computer so that changes are implemented.
If there are any problems run the program IBTEST to check the board and refer to the owners manual.

Sometimes the DA Parameters need some special attention when a complete replacement of the system folder has been made.
a) Copy the DA param folder.
b) Run the current version of the Data Acquisition program.
c) Select each of the parameters and fix each of the requested numbers. Sometimes this will cause an error massage. At the click of the mouse the error massage will disappear and a parameter window will open. Make the changes on it at that point.
d) Save every change. Next time the parameter is opened there will be no problem.