STUDIES OF PROTEIN-PROTEIN AND PROTEIN-WATER INTERACTIONS BY SMALL ANGLE X-RAY SCATTERING, TERAHERTZ SPECTROSCOPY, ASMOS, AND COMPUTER SIMULATION

BY

SEUNG JOONG KIM

B.S., Seoul National University, 2003
M.S., University of Illinois at Urbana-Champaign, 2004

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Physics
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Professor Taekjip Ha, Chair
Professor Martin Gruebele, Director of Research
Professor David Ceperley
Professor Oleksii Aksimentiev

Abstract

The protein folding problem has been one of the most challenging subjects in biological physics due to its complexity. Energy landscape theory based on statistical mechanics provides a thermodynamic interpretation of the protein folding process. We have been working to answer fundamental questions about protein-protein and protein-water interactions, which are very important for describing the energy landscape surface of proteins correctly.

At first, we present a new method for computing protein-protein interaction potentials of solvated proteins directly from SAXS data. An ensemble of proteins was modeled by Metropolis Monte Carlo and Molecular Dynamics simulations, and the global X-ray scattering of the whole model ensemble was computed at each snapshot of the simulation. The interaction potential model was optimized and iterated by a Levenberg-Marquardt algorithm.

Secondly, we report that terahertz spectroscopy directly probes hydration dynamics around proteins and determines the size of the dynamical hydration shell. We also present the sequence and pH-dependence of the hydration shell and the effect of the hydrophobicity. On the other hand, kinetic terahertz absorption (KITA) spectroscopy is introduced to study the refolding kinetics of ubiquitin and its mutants. KITA results are compared to small angle X-ray scattering, tryptophan fluorescence, and circular dichroism results. We propose that KITA monitors the rearrangement of hydrogen bonding during secondary structure formation.

Finally, we present development of the automated single molecule operating system (ASMOS) for a high throughput single molecule detector, which levitates a single protein molecule in a 10 μm diameter droplet by the laser guidance. I also have performed supporting calculations and simulations with my own program codes.

For my forever friend, In-ha (1977-2003),

To my brother and parents,

and

To my lovely wife, Su Yeon

내 영원한 벗,
김 인하 (1977-2003)를 기리며,

언제나 자랑스러운
내 동생과 부모님께,

그리고 사랑하는
내 인생의 반쪽, 수연이 에게

Acknowledgments

Most of all, I would like to thank my advisor, Martin Gruebele for everything he has done to influence me since 2004. He always inspires me with new ideas and gives the best advice in any circumstance - he is a real mentor and great scientist! I have learned numerous scientific methods, how to manage my time and schedule, and even how to collaborate with people. I am indebted so many things to him.

I would like to thank Charles Dumont and Yoshitaka Matsumura for assistance during SAXS data collection, and Dr. Elena Kondrashkina and Dr. Liang Guo at Argonne National Laboratory for helpful assistance in setting up the beam line for SAXS experiments. The SAXS work in Chapter 2 was supported by National Science Foundation grant MCB 0613643. Time on BioCAT-18, managed by Prof. Thomas Irving, was supported by proposal GUP-6360 of the Advanced Photon Source at Argonne National Laboratory.

I especially appreciate Simon Ebbinghaus and Benjamin Born for the cordial welcome and the sharing of their THz instruments and experimental results when I visited Ruhr-University-Bochum, Germany. I was very lucky to have the chance to collaborate with these nice people. I wish to thank Professor Martina Havenith and Professor David Leitner for their excellent supervision in Germany. I wish to thank E. Bründermann and G. Schwaab for the initial instrument development and many helpful discussions, and software development. I thank Matthias Krüger for programming the THz data acquisition software. I gratefully acknowledge financial support from the Human Frontier Science Program for Chapter 3. The KITA work in Chapter 4 was also supported by a grant MCB-0613643 of the National Science Foundation.

I especially would like to thank Krishnarjun Sarkar, my colleague in our group, for the single molecule work in Chapter 5 and 6. He is a model of passion and diligence. I wish express my gratitude to Dr. J. D. McDonald for his excellent scientific advice and

the design of Data Acquisition Box.

Edgar Larios, the former Gruebele group member, has taught me many basic biochemical techniques for the preparation of protein samples. I wish he would succeed in his new carrier. In addition, I would like to thank the committee of Professors Taekjip Ha, David Ceperley, and Oleksii Aksimentiev for overseeing my graduate study.

The four years in the Gruebele group was a big turning point in my life, and I enjoyed it a lot. The moments with so nice people in the Gruebele group are going to be unforgettable forever. Feng Liu, Sharlene Denos, Krishnarjun Sarkar, Daniel Weidinger, Charles Dumont, Apratim Dhar, Praveen Chowdary, Erin Carmichael, Greg Scott, Sumit Ashtekar, and Praveen Sundaradevan, I appreciate you guys, go Gruebele group!

My dissertation is dedicated to the memory of my forever friend, In-ha Kim, who has passed away from a heart attack at 2003. He always has dreamed to be a great astronomer. I'm sure his spirit is now smiling sweetly at the completion of my graduate study. I will never forget so many great memories with him, since my high school.

Luckily I grew up in a family of warm affection. I'd like to thank my brother and parents for their concern and constant supports. Last, but not least, my special thanks go to my wife, Su Yeon Kim. Most importantly, none of this would have been possible without her love and encouragement. More than two years of long-distance driving and phone calls finally pay off. We're so excited to be able to live together in San Francisco bay area.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ANS	1-anilino-8-naphthalene sulfonate
APS	Advanced Photon Source
ASMOS	Automated Single Molecule Operating System
CCD	Charge-coupled device
CD	Circular Dichroism
CPU	Central Processing Unit
CS	Critical Section
Da	Dalton (=gram / mole)
DAB	Data Acquisition Box
DAQ	Data Acquisition
DCE	Data Communications Equipment
DIO	Digital Input/Output
DTE	Data Terminal Equipment
EDTA	ethylenediaminetetraacetic acid
FRET	Förster resonance energy transfer
FTIR	Fourier Transform Infrared spectroscopy
FWHM	Full width at half maximum
GLMT	Generalized Lorentz-Mie theory
GuHCl	Guanidine Hydrochloride
GUI	Graphical user interface
IPTG	Isopropyl β-D-1-thiogalactopyranoside
IR	Infrared
KITA	Kinetic Terahertz Absorption
λ^{*}_{6-85}	Lambda repressor fragment 6-85 with a tryptophan at residue 22
LM	Levenberg-Marquardt
LMD	Langevin molecular dynamics
LMT	Lorenz-Mie Theory
MD	molecular dynamics

μM	micro Molar
mM	milli Molar
MMC	Metropolis Monte Carlo
MW	Molecular Weight
MWCO	Molecular Weight Cutoff
NALMA	N-acetyl-leucine-methylamide
NI	National Instruments
nm	nano meter
NMR	Nuclear Magnetic Resonance
PCI	Peripheral Component Interconnect
PDB	Protein Data Bank
PFI	Programmable Function Interface
PMT	Photomultiplier tube
RH	Relative Humidity
RAID	Redundant Array of Independent/Inexpensive Disks
RTSI	Real-Time System Integration
SATA	Serial Advanced Technology Attachment
SAXS	Small Angle X-ray Scattering
SDS	Sodium Dodecyl Sulfate
SDS-PAGE	Sodium Dodecyl Sulfate - Polyacrylamide Gel Electrophoresis
THz	Terahertz
THz-TDS	Terahertz Time Domain Spectroscopy
TSQ	Thread Safety Queue
Ub	Ubiquitin
Ub*	Ubiquitin with a tryptophan at residue 45
UV	Ultraviolet
VMD	Visual Molecular Dynamics
WAXS	Wide Angle X-ray Scattering

Publications

Parts of the work presented in this dissertation are based on the following publications to which the author significantly contributed.

In Chapter 2,

- Kim, S.J., C. Dumont, and M. Gruebele, Simulation-base fitting of protein-protein interaction potentials to SAXS experiments. Biophysical Journal, 2008(94): p. 4924-4931.

In Chapter 3,

- Ebbinghaus, S., S.J. Kim, M. Heyden, X. Yu, U. Heugen, M. Gruebele, D. Leitner, and M. Havenith, *An extended dynamical hydration shell around proteins.* Proc. Nat. Acad. Sci. USA, 2007. **104**: p. 20749-20752.

- Ebbinghaus, S., S.J. Kim, M. Heyden, X. Yu, M. Gruebele, D. Leitner, and M. Havenith, *Protein sequence- and pH-dependent hydration probed by Terahertz spectroscopy.* J. AM. Chem. Soc., 2008. **130**(8): p. 2374-2375.

- Born, B., S.J. Kim, M. Gruebele, and M. Havenith, The Terahertz dance of water with the proteins: The effect of protein flexibility on the dynamical hydration shell of ubiquitin. Faraday Discussion 141: Water – From Interfaces to the Bulk, 2008. **in press**.

In Chapter 4,

- Kim, S.J., B. Born, M. Havenith, and M. Gruebele, *Real-time detection of protein-water dynamics upon protein folding by Terahertz absorption spectroscopy.* Angewandte Chemie International Edition, 2008. 47(34): p. 6486-6489 (as an **Inside cover story at p. 6302**).

In Appendix A,

- Dumont, C., Y. Matsumura, S.J. Kim, J. Li, E. Kondrashkina, H. Kihara, and M. Gruebele, *Solvent-tuning collapse and helix formation time scales of lambda6-85.* Protein Science, 2006. **15**: p. 2596-2604.

Chapter 1 Introduction

1.1 The protein folding problem

Proteins are polymers made of amino acids, joined together by peptide bonds. Each protein has a specific sequence of amino acids, which is uniquely encoded in the genetic code. Through the process of transcription and translation via RNA, genetic codes can be used for the construction of proteins.

$$\text{DNA} \rightarrow \text{RNA} \xrightarrow{Ribosome} \text{Peptide chain} \rightarrow \text{Folded Protein} \qquad (1.1)$$

Proteins are essential to the structure and function of living organisms. Many proteins perform a wide variety of biological functions, such as catalysis of chemical reactions, while some other proteins play structural or mechanical roles. Also, functions of many other proteins involve oxygen transport, immune function, muscle contraction, etc.[1]

In order to perform specific biological functions, proteins must have a particular native structure, the folded state. If a protein lacks the correct structure, it might be inactive, functionless or misfolded and malfunctioning. Misfolded protein or aggregation of protein causes thousands of diseases such as Parkinson's disease and Alzheimer's.[2] The transformation from an inactive, denatured (unfolded) state to the native (folded) state is called the "protein folding". The amino acid sequence encoded in DNA determines how fast the protein folds and what structures the native protein will have eventually.[3]

However, how a protein folds still remains one of the most challenging problems in biology and biological physics [4], because it is a highly complex process, having almost infinite numbers of conformations. In early 1960s, Anfinsen et al. published a pioneering work on the folding kinetics of RNase.[5] Cyrus Levinthal pointed out in 1968 that protein folding cannot be a random process because that would require too

large a time scale, so there must be shortcuts for the folding process.[6, 7] In order to answer the fundamental questions of the protein folding, a lot of researchers have been working to combine the theory and the experimental results,[8, 9] and the energy landscape theory has been established since 1990s with great success. [10, 11]

1.2 Energy landscape theory

The energy landscape theory based on statistical mechanics provides a thermodynamic interpretation of the protein folding process. Thermodynamically, the protein folding process can be described as an energy landscape that looks like a funnel [10, 11], as briefly shown in [Figure 1.1].

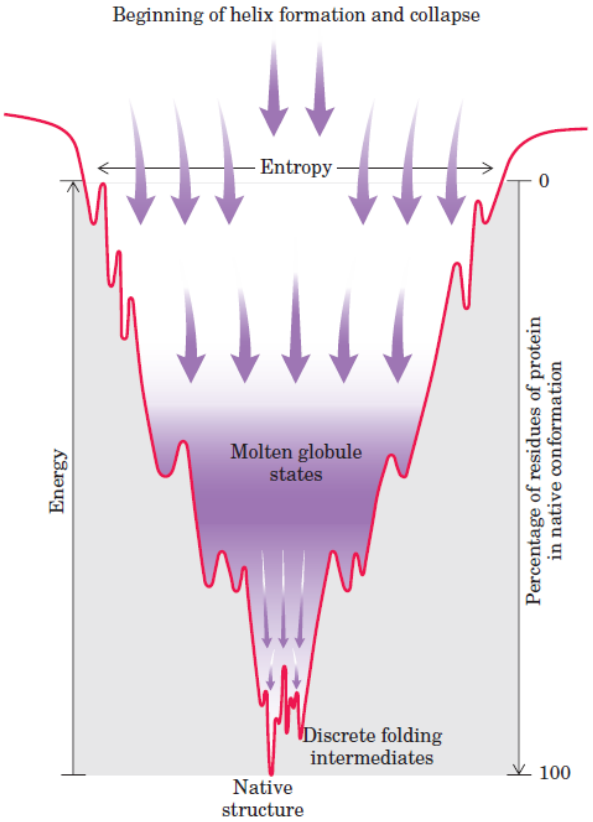


Figure 1.1 : The funnel-like energy landscape model

Nelson, D.L. and M.M. Cox, *Lehninger Principles of Biochemistry*. 4th ed. 2004. p149.

The horizontal axis in [Figure 1.1] describes conformational entropy of the protein structure, while the vertical axis describes the level of energy, enthalpy. Another vertical element, Q corresponds to the "percentage of residues of protein in the native conformation". The unfolded states (Q~0%) are characterized by a high degree of conformational entropy and energy. As the folding proceeds, the narrowing of the funnel represents a decrease in the number of conformations, and hence in entropy. Small depressions along the sides of the energy funnel represent semi-stable intermediates that can briefly slow down the folding process.[1] At the bottom of the funnel, the protein finally reaches the native, folded state (Q=100%) characterized by the lowest energy with minimum conformational entropy.

1.3 Various experimental techniques for the protein folding problem

A number of experimental techniques have been developed to describe the energy landscape surface of the proteins in their probe-dependent coordinates. So far, the current experimental techniques can be classified into the category of equilibrium and relaxation techniques. Equilibrium techniques include NMR, while relaxation includes a fast mixing (or a stopped-flow), temperature jump, pressure jump, etc.[8]

Combined with those techniques, Small Angle X-ray Scattering (SAXS), Circular Dichroism (CD), Terahertz spectroscopy, and Single Molecule Detection are essential probes for the investigation of the protein folding problem. We can measure the radius of gyration of proteins by SAXS [12], which can be applied to study the time-resolved kinetics of the protein folding.[13] SAXS kinetics combined with stopped-flow apparatus measures the radius of gyration to determine how fast proteins and their mutants collapse. Circular dichroism is a useful technique to monitor secondary structure formation, such as alpha helix and beta sheet, by measuring the difference of clockwise and counterclockwise circular polarizations.[14] Moreover, Terahertz spectroscopy is a perfect probe for investigating collective motions such as conformational changes and the

formation of hydration shells around the protein [13-15], while single molecule fluorescence is used to study the conformational change of the protein on the basis of a single molecule.[15, 16]

1.4 Outline

I have been working with Dr. Martin Gruebele to answer fundamental questions about protein-protein and protein-water interactions, which are very important for describing the energy landscape surface of the proteins correctly. My doctoral research has been balanced between experimental and computational work at the interdisciplinary interface of physics, chemistry and biology.

In Chapter 2, we present a new method for computing protein-protein interaction potentials of solvated proteins directly from SAXS data. I have regularly visited BioCat-18 of the Advanced Photon Source at Argonne National Lab, IL to perform SAXS experiments. An ensemble of proteins was modeled by Metropolis Monte Carlo and Molecular Dynamics simulations, and the global X-ray scattering of the whole model ensemble was computed at each snapshot of the simulation. The interaction potential model was optimized and iterated by a Levenberg-Marquardt algorithm (Biophysical Journal, 2008).

In Chapter 3, we report that terahertz spectroscopy directly probes hydration dynamics around proteins and determines the size of the dynamical hydration shell. In collaboration with the Havenith group in Ruhr-University-Bochum, Germany, we measured a non-linear concentration dependence of the Terahertz absorption coefficient of protein solutions, which indicated the overlap of hydration shells and how far hydration shells could eventually extend from the protein surface. This interesting behavior was in strong agreement with molecular dynamics simulations which showed that the dynamics of water molecules are affected by the protein at a distance out to ~10Å from the protein surface. (PNAS, 2007) Also we probed the sequence and pH-

dependence of hydration shells (JACS, 2008) and presented that exposed hydrophobic residues significantly affects the formation of the dynamical hydration shells. (Faraday Discussion, 2008, in press)

In Chapter 4, kinetic terahertz absorption (KITA) spectroscopy is introduced to study folding of solvated biomolecules. Also in collaboration with the Havenith group, we applied KITA to the refolding kinetics of ubiquitin and of three side chain truncation mutants designed to disrupt the hydrophobic core and increase overall protein flexibility. KITA results are compared to small angle X-ray scattering, tryptophan fluorescence, and circular dichroism results. The KITA signal rapidly relaxes to the native protein's value, on the same millisecond time scale on which secondary structure formation is detected by circular dichroism. Both processes are much faster than acquisition of native-like fluorescence. We propose that KITA monitors the rearrangement of hydrogen bonding during secondary structure formation, and suggest future experimental tests and applications to folding dynamics with this new technique. (Angewandte Chemie, 2008, in press as a cover story)

Most of protein folding measurements have been conducted on the basis of bulk samples up to now. What we get in a bulk system is just a statistical average of a protein ensemble. In Chapter 5, I describe development of the automated single molecule operating system (ASMOS) for a high throughput single molecule detector, levitating a single protein molecule in a 10 μm diameter droplet by the laser guidance. The highly automated data acquisition module provides fluorescence lifetime and photon spacing information for large numbers of single proteins, leading to the single molecule statistical analysis. To improve the efficiency of single molecule detection, in Chapter 6, I performed supporting calculations and simulations of the laser light scattering by a small droplet as well as radiation forces in laser guidance based on Generalized Lorentz-Mie theory (GLMT). The evaporation effect of a small droplet was added on the basis of fundamental statistical mechanics. These calculations and simulations are performed with my own program codes in C language.

Chapter 2 Simulation-based fitting of protein-protein interaction potentials to SAXS experiments

2.1 Abstract

We present a new method for computing interaction potentials of solvated proteins directly from small angle X-ray scattering data. An ensemble of proteins is modeled by Monte Carlo or molecular dynamics simulation. The global X-ray scattering of the whole model ensemble is then computed at each snapshot of the simulation, and averaged to obtain the X-ray scattering intensity. Finally, the interaction potential parameters are adjusted by an optimization algorithm, and the procedure is iterated until the best agreement between simulation and experiment is obtained. This new approach obviates the need for approximations that must be made in simplified analytical models. We apply the method to lambda repressor fragment 6-85 (λ^*_{6-85}) and *fyn*-SH3. With the increased availability of fast computer clusters, Monte Carlo and molecular dynamics analysis using residue-level or even atomistic potentials may soon become feasible.

2.2 Introduction

Small angle X-ray scattering (SAXS) is a convenient tool for determining protein-protein interaction potentials in solution. A major driving force of this work has been the need for determining ideal conditions for protein crystallization. Thus, the focus has been on the effect of the concentration of precipitation agents and co-solvents [17, 18].

Two additional areas could benefit greatly from the effective potentials provided by SAXS studies. One is the study of hydration shells around proteins. Neutron scattering, NMR spectroscopy, simulation, and terahertz spectroscopy have shown that

solvent shells of substantial thickness exist around proteins [19-22]. Dynamical hydration effects studied by terahertz spectroscopy extend to > 10 Å from the protein surface [19]. Protein-protein interactions are mediated by such solvent shells, and thus contain information about the solvent shells when measured at sufficiently high concentrations. The other area is the study of transient protein aggregation. Very rapidly folding proteins have folding time scales comparable to the lifetime of transient aggregates [20, 21]. Such transient aggregates can nucleate irreversible aggregation [22, 23], a process linked with numerous diseases. Protein-protein interaction potentials play a key role in defining how easily such nuclei form.

Effective interaction potentials are currently extracted from SAXS data with the aid of analytical approximations to speed up the calculation [17]. The random phase approximation treats each protein molecule as an independent scatterer characterized by a form factor. The form factor can be obtained approximately by extrapolating SAXS measurements to infinite dilution [24]. The observed scattering intensity is then assumed to be a product of the form factor and of a scattering factor, an approximation strictly valid over the full range of scattering angles only for dilute particles. From the scattering factor, a radial pair distribution function and corresponding radial effective potential are obtained. Square well, and exponential potentials are used because they have simple Fourier transforms [18]. The commonly used DLVO form consists of a hard sphere cutoff, and two Yukawa potentials ($\sim \exp[-(r-r_0)/\delta] / r$) for long-range repulsion and short range attraction between proteins.

Increases in computing power enable a more direct approach, which we introduce here. Simulation of multi-protein ensemble dynamics is followed by evaluation of the X-ray scattering of the whole ensemble. Iteration can then be used to refine force fields “on the fly” without any low-concentration approximations or scattering analysis approximations.

[Figure 2.1] outlines our approach. We first simulate the dynamics of an ensemble of dozens to hundreds of model proteins that interact via an adjustable interaction potential. Either Monte Carlo or molecular dynamics simulations are used to sample configurations of the ensemble. We then calculate the global X-ray scattering

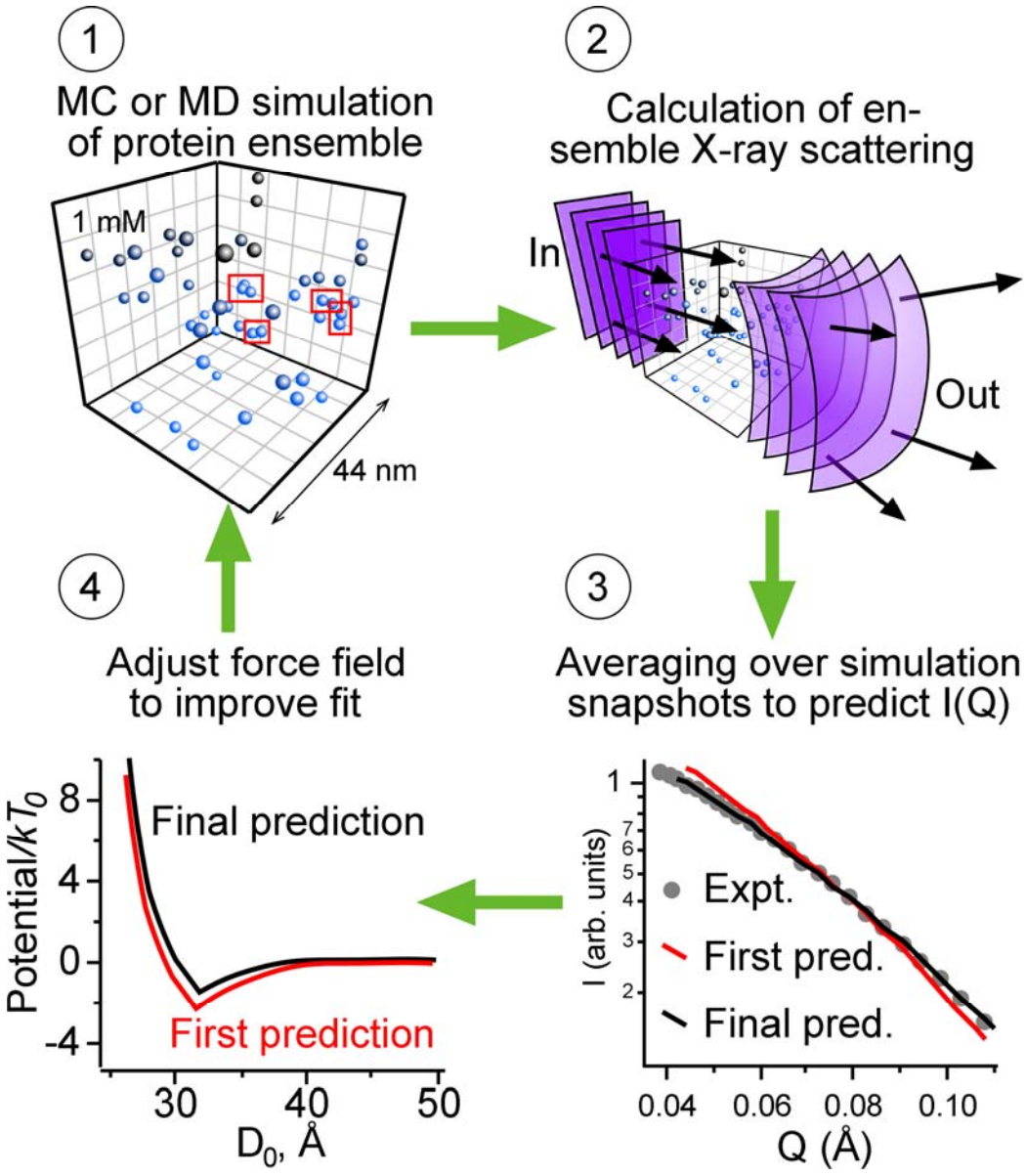


Figure 2.1 : Method for extracting the protein-protein potential from SAXS data

In step 1, a protein ensemble of up to 100 molecules is simulated by Monte Carlo or Molecular Dynamics. In step 2, the exact X-ray scattering for the model ensemble is evaluated at each simulation snapshot. In step 3, the average X-ray scattering curve is obtained and compared with experimental data. In step 4, the interaction potential is adjusted by steepest descent for the next round of simulation.

intensity of the entire model ensemble at each configuration, eliminating the need for low concentration or random phase approximations. The resulting series of scattering intensities is averaged to obtain the steady-state SAXS intensity as a function of scattering angle. An optimization algorithm compares the computed signal with the experimental signal, and modifies the adjustable interaction potential for the next round

of simulation. The process repeats iteratively until the experimental data is matched with the smallest least-squares deviation. Any form of potential can thus be fitted exactly for polydisperse model particles at any concentration.

In this first application, we determine isotropic interaction potentials, and hence assume spherical model protein monomers. Aggregates can have any shape made from these monomer building blocks, up to the size of the box used for simulation, typically 20 monomer diameters or more. Thus the analysis must be truncated at large scattering angles, but it does not assume spherical aggregates or low monomer concentration. We illustrate the method by fitting experimental data for the two proteins λ^*_{6-85} and *fyn*-SH3 to several potential models. The ethylene glycol-water solvent we use is similar to the one used in recent SAXS studies of folding kinetics [13]. With the advent of interaction potentials based on sums of amino acid pair-interactions, the simulation-direct fitting approach could yield anisotropic interaction potentials in the near future, revealing potential aggregation sites, or local changes in the protein hydration shell.

2.3 Materials and methods

2.3.1 Proteins

The wild type of λ repressor is a DNA-binding phage regulatory protein, which controls the lambda switch in bacterial cells. The small engineered lambda repressor fragments, λ^*_{6-85} is an 80-residue, five-helix globular protein of molecular mass 9.2 kDa [inset in Figure 2.5A]. The protein we used in SAXS experiments contained the mutations Tyr22Trp, Glu33Tyr, Gly46Ala, and Gly48Ala, engineered by site-directed mutagenesis (Stratagene Quickchange kit, La Jolla, CA) based on a wild-type plasmid donated by Terry Oas [25]. *fyn*-SH3 is a predominantly β-sheet protein (Molecular mass 9.3 kDa) [inset in Figure 2.5B] with 78 residues and a tag of 6 histidine residues. The sequence, donated by Alan Davidson, has mutations Val1Ser, Val5Glu, Ala39Val, and Val55Phe [26].

Genes for the two proteins were inserted into the PET-15b vector, expressed in Rosetta TM (DE3) pLysS cells (Novagen, San Diego, CA), and grown in LB broth at 37 °C for 8 hours. After induction with IPTG (isopropyl-β-D-thiogalactopyranoside) at 25 °C for 12 hours, cells were lysed with a French press, and the supernatant was collected after centrifugation. Proteins were selectively bound to a nickel-agarose His-tag binding column (Pharmacia) and eluted with a 250 mM imidazole buffer. The 6-Histidine tag of λ^*_{6-85} was cleaved by thrombin (VWR), and additional purification was performed with Amicon 3 kDa and 30 kDa membranes (Fisher Scientific). fyn-SH3 was used with the His tag. The identity of λ^*_{6-85} and fyn-SH3 was confirmed by electrospray ionization mass spectroscopy and their purity by sodium dodecyl sulfate polyacrylamide gel electrophoresis.

Final protein concentrations in buffers used for experiments were determined by near-UV absorption spectroscopy at 280 nm of the tryptophan and tyrosine residues as described by Edelhoch [27]. We have found this procedure to yield similar results in aqueous and aqueous-osmolyte buffers. We estimate a relative accuracy of ±1% for dilution series from the same sample, and an absolute accuracy of about ±5%. Results are rounded to the nearest 10 μM.

2.3.2 SAXS measurements

SAXS measurements were performed at the Biophysics Collaborative Access Team Beamline of the Advanced Photon Source at Argonne National Laboratory (Argonne, IL) [28]. As shown in [Figure 2.2], an Aviex CCD camera with an active area of approximately 160×80 mm^2 (2084×1042 pixels, 78 μm gap between pixels), located 1.9 m from sample, was used to collect data in the scattering angle range of $Q = 4\pi \sin\theta / \lambda = 0.03\text{-}0.12$ Å^{-1}, at a nominal wavelength of 1 Å. Low concentration data for *fyn*-SH3 were also collected with a Pilatus CCD camera. The X-ray beam was collimated to a spot size of 300×130 μm^2 at the sample cell. See [Appendix A] for more information in detail.

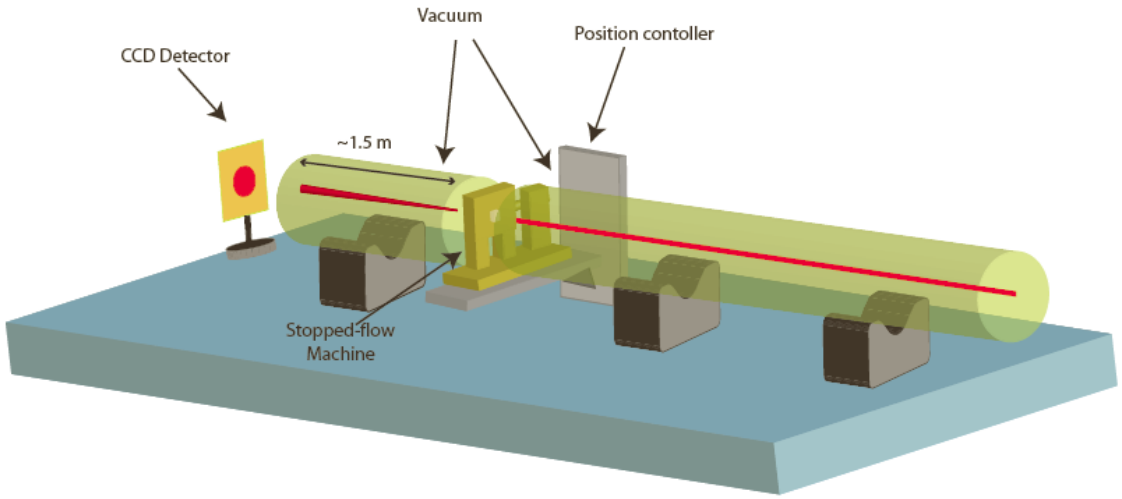


Figure 2.2 : SAXS setup on BioCAT-18 of APS at Argonne National Laboratory.[1]

From ref. [29]: Larios, E., *a Computational-Experimental Study of Small Globular Proteins*, in *Physics Ph.D. Thesis*. 2005, University of Illinois at Urbana-Champaign.

To reduce radiation damage, and to enable a direct comparison with our previous SAXS folding study of λ^*_{6-85}, we performed our experiments in a 45:55 vol. % ethylene glycol/water buffer. The ionic strength was 50 mM phosphate at pH 7.0. The temperature in all experiments was –28 ± 1 °C, cooled by a Neslab ULT-80DD recirculator. Steady-state SAXS data were collected in a UNISOKU sample cell with 80 µl volume and 50 µm thick sapphire windows. The exposure time was 500 ms for λ^*_{6-85}, and 300 ms for *fyn*-SH3 (4 frames of 200 ms on the Pilatus detector), based on extensive exposure/concentration tests for protein damage. We measured steady-state SAXS data of λ^*_{6-85} up to 2.92 mM, and of *fyn*-SH3 up to 1.68 mM, without any visible aggregation at room temperature or at –28 °C. Each sample was filtered with a 0.2 µM pore syringe filter (Corning) before use. The raw data were angle-averaged with logarithmic weighting in Q, and a reference buffer curve was subtracted.

[1] By courtesy of Edgar Larios, an alumnus of the Gruebele group

2.3.3 Interaction potentials

To enable Monte Carlo or molecular dynamics simulations, a protein-protein interaction potential has to be chosen. We tested several pair wise-additive isotropic interaction potentials not easily fitted by analytical methods. At short distance an r^{12} repulsive term was used instead of the commonplace hard sphere wall:

$$U_L = \varepsilon\left[\left(\frac{D_0}{r}\right)^{12} - 2\right] \qquad (r < D_0)\,. \tag{2.1}$$

Past the potential minimum at D_0, exponential, Gaussian and Yukawa forms were used in various combinations to model both attractive and repulsive-attractive potentials:

$$\left.\begin{aligned} U_E &= -\varepsilon \exp\left[-\left(\frac{r - D_0}{\delta}\right)\right] \\ U_G &= -\varepsilon \exp\left[-\left(\frac{r - D_0}{\delta}\right)^2\right] \\ U_Y &= -\varepsilon\left(\frac{D_0}{r}\right)\exp\left[-\left(\frac{r - D_0}{\delta}\right)\right] \end{aligned}\right\} (r > D_0) \tag{2.2}$$

, where ε is the potential depth, D_0 is the center-of-mass distance between proteins where the repulsive potential wall begins, and δ is the attractive potential range.

The softer than hard-sphere potential wall, not easily amenable to the analytical treatment, highlights the fact that no reference potential assumptions need to be made. In our first application, we assumed isotropically interacting particles and pair wise additive potentials, although nonspherical particles and n-body potentials could be implemented in the future because our approach requires only that the total potential energy for the multi-protein system can be evaluated.

2.3.4 Configurational sampling

To avoid the need for low-concentration approximations, we sample a whole protein ensemble much larger than the typical aggregate size. Protein configurations were sampled by two methods: Metropolis Monte Carlo sampling (MMC), which illustrates computation of the scattering curve from a thermal simulation, and Langevin molecular dynamics (LMD), to illustrate computation of scattering curves from real-time dynamics simulations. In both approaches, we distributed n=25 to 100 spherically symmetric model protein particles in a spherical or cube-shaped volume, the latter with periodic boundary conditions. The diameter of the simulation volume was determined by the experimental protein concentration. To reduce oscillatory boundary artifacts in the SAXS calculation, the diameter of the volume was varied randomly about the average. Test runs with up to 20,000 protein particles confirmed that full convergence over the desired range of Q could be achieved rapidly with 25 particles for *fyn*-SH3 and with 100 particles for λ^*_{6-85} over the full experimental concentration range.

For MMC sampling, we started out with a random distribution of particles. Single particles were then chosen at random, and moved by random displacements inside the spherical volume. Each move was accepted or rejected based on the Metropolis criterion by computing the change in total energy, ΔE [30]. When the net energy change was negative, the move was accepted, while a positive energy change was accepted with a probability of $\exp(-\Delta E/k_B T)$. Equilibration of the total energy to within the statistical noise typically required $50n$ moves for λ^*_{6-85}. This sampling was repeated until the scattering intensity (see below) was a smooth function of Q. The longest runs provide estimated error bounds for the computed scattering curve.

For molecular dynamics sampling in real time, we used a Langevin-MD approach in a cubic volume with periodic boundary conditions. Each protein particle was subject to a vectorial force resulting from the other protein particles, and to a random Brownian force simulating the implicit solvent dynamics. In addition, the Brownian motion was countered by a vectorial damping term. Inertial forces were neglected, resulting in $3n$ equations of motion

$$-\frac{\partial V}{\partial r_{i,m}} - \gamma \frac{dr_{i,m}}{dt} + \xi_{i,m}(t) = 0. \tag{2.3}$$

For non-spherical particles subject to anisotropic interaction potentials, an additional set of 3*n* equations for rotational diffusion would have to be solved, but no additional complications are introduced by our approach. In [Equation (2.3)], V is the interaction potential summed over all protein pairs ([Equation (2.1)] and [Equation (2.2)]). Protein particle m is at position $\mathbf{r}_m = (r_{x,m},\ r_{y,m},\ r_{z,m})$. $\gamma = kT / D(T,P)$ denotes the velocity relaxation rate, which depends on the diffusion coefficient D, assumed independent of coordinate. $\xi_i(t)$ is Gaussian white noise with zero mean, and a variance set to satisfy the Onsager fluctuation-dissipation theorem that relates ξ and γ [31]. The equations of motion were integrated by a standard integrator using finite-difference derivatives (thus Brownian noise or discontinuities in the potential derivative are not a problem). Derivatives with respect to a single particle, like the energy change ΔE, could be evaluated efficiently. The protein distribution was allowed to evolve to a mean particle deviation of at least 3.4 R_g before sampling the next configuration, to ensure that the scattering calculation did not needlessly sample very similar configurations.

2.3.5 Scattering signal

For each multi-protein configuration from the MMC or LMD simulations, we calculated the exact X-ray scattering by evaluating

$$F_{\text{total}}(\mathbf{q}) = \sum_{m=1}^{n} F_m e^{-i\mathbf{q}\cdot\mathbf{r}_m} , \tag{2.4}$$

where F_{m} is the scattering amplitude for particle m. Because we are determining isotropic interaction potentials here, we approximated each protein particle by a sphere and used the corresponding F_{m} [32, 33]. The assumption of individual spherical particles sets an upper limit on the Q values that can be fitted. A more realistic electron distribution based on diffraction data would have to be used if anisotropic potentials and large Q values are to be used in fitting. [Equation (2.4)] treats the scattering of the model protein ensemble exactly at any concentration and for any aggregation state that is small compared to the size of the simulation box. Thus, no extrapolations to dilute samples or analytical approximations usually needed for polydisperse systems need to be made. The total scattering intensity is obtained from

$$I(\mathbf{q}) = \left| F_{\text{total}}(\mathbf{q}) \right|^2 , \tag{2.5}$$

and averaged over all configurations sampled by the simulations to yields the average SAXS scattering intensity $I(Q)$ for direct comparison with experiment.

Approximately 100,000 configurations were averaged for each concentration to obtain a smooth $I(Q)$ for comparison with experiment. To minimize boundary effects and oscillations of the intensity at low Q below the experimental noise level, either a spherical volume was chosen, and its volume was changed randomly about the average value required by each protein concentration [34], or an spherical volume from the center of a periodic boundary condition box was chosen for the X-ray scattering calculation.

2.3.6 Data fitting

We fitted three potential parameters: potential depth ε, potential range δ and potential wall $D_0 \equiv 2R_0$. An efficient Levenberg-Marquardt optimization algorithm [35, 36] was applied to fit the potential parameters to the experimental concentration-dependent scattering data. Minimal evaluation of $I(Q)$ is desirable because each concentration point requires a large number of MMC/LMD simulations to yield a smooth curve.

We also fitted a fourth parameter, the radius of gyration R_g of the model proteins, to account for the direct effect of particle size on the scattering data. R_0 measures monomer size from the point of view of the interaction potential, while R_g measures monomer size from the point of view of the scattering intensity. R_g is not entirely independent of R_0. For an ideal hard sphere monomer, $R_g / R_0 = \sqrt{3/5}$. Deviations from spherical shape, and a tapering of the electron density distribution due to hydration or a soft potential wall [Equation (2.1)], both are effectively accounted for by allowing deviations from this ratio. A large deviation would indicate that a better model for the monomeric proteins is needed.

2.4 Results

2.4.1 Concentration-dependent SAXS of λ^*_{6-85}

[Figure 2.3] shows the concentration dependence of the scattering intensity as a function of Q for the λ^*_{6-85} Q33Y G46A G48A mutant. A Guinier plot ($\ln(I)$ *vs.* Q^2, not shown) deviates from linearity below Q^2 = 0.006 Å^{-2}, indicating some aggregation. Dilution of samples shows that this aggregation is reversible over the concentration range we studied. No deviations were observed at concentrations below 100 μM or Q up to 0.11 Å^{-1}, indicating that the spherical approximation for protein monomers is good for λ^*_{6-85} over the range of scattering angles considered here.

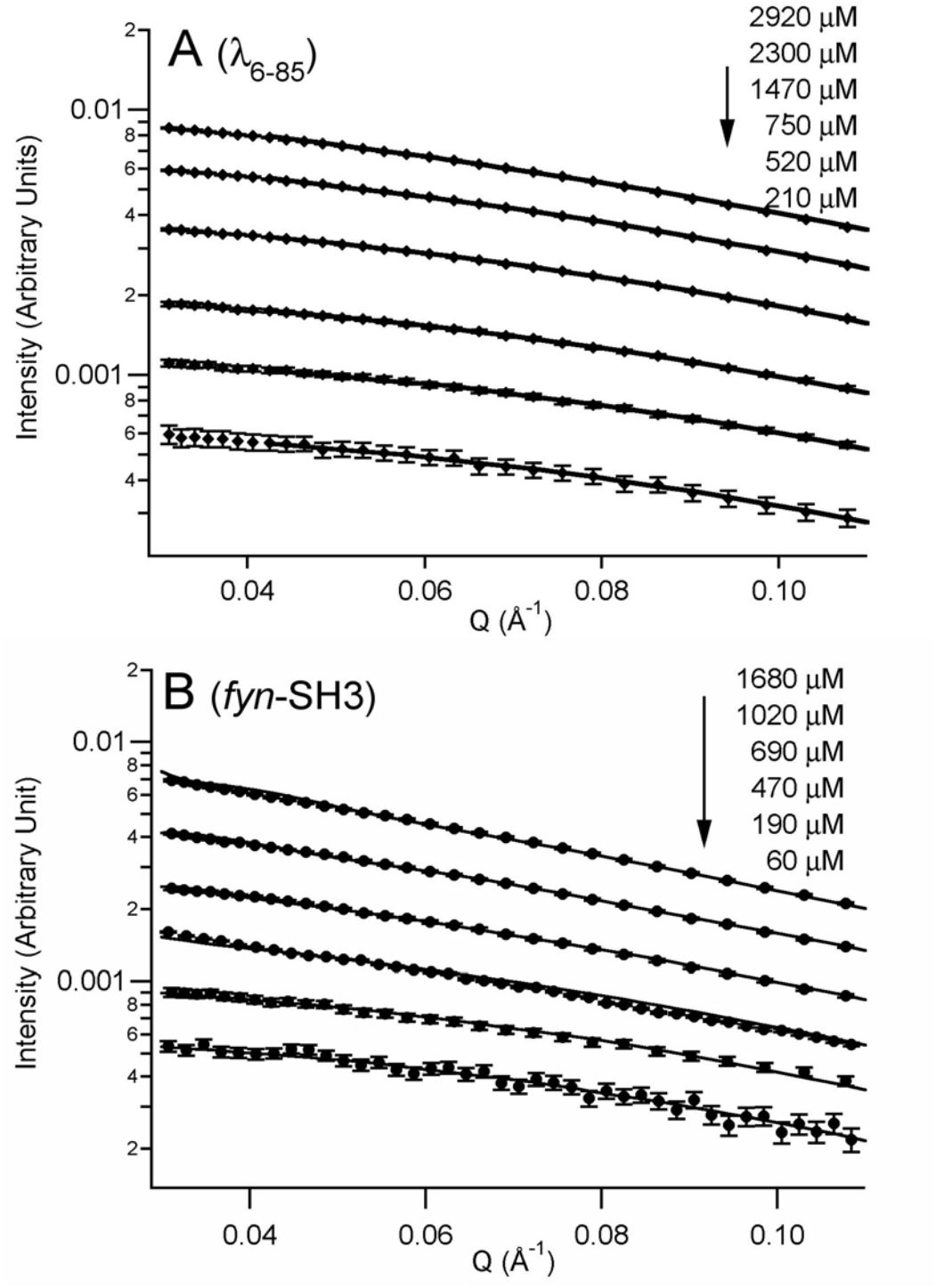


Figure 2.3 : SAXS data and MMC simulation results

Scattering intensity *vs.* magnitude of the scattering vector For λ^*_{6-85} (A) and *fyn*-SH3 (B). The lines going through the experimental data points are fits from MMC (Metropolis Monte Carlo) simulation.

2.4.2 MMC fitting results for λ^*_{6-85}

Simulations were performed by the MMC method. The best fit to experimental data [Figure 2.3A] was obtained with a U_L+U_E potential (Lennard-Jones wall (r^{12} repulsive) + exponential attractive).

$$U = \begin{cases} \varepsilon\left[\left(\frac{D_0}{r}\right)^{12} - 2\right] & (r < D_0) \\ -\varepsilon \exp\left[-\left(\frac{r - D_0}{\delta}\right)\right] & (r > D_0) \end{cases} \tag{2.6}$$

The calculated radius of gyration is 13.52 Å, the potential depth is 1.5 kT_0, with 3.6 Å of potential range, and the potential wall beginning at D_0≈31.8 Å ([Table 2.1] and [Figure 2.4A]; T_0 = 245 K). A total of 100 proteins were used in 5,000 Metropolis iterations to obtain equilibrated results for each configuration, and 100,000 configurations were sampled. As one might expect, two parameters of this fit are somewhat correlated, the potential range and depth.

Table 2.1 : Best fit of the λ^*_{6-85} SAXS data to a U_L + U_E (r^{12} repulsive + exponential attractive) potential
Also shown are the root mean square errors (RMSE) for the best fit at individual concentrations. All RMSEs of the fit lie within the experimental error. kT_0 corresponds to 245 K.

Potential type	R_g (Å)	**Potential depth** ε (kT_0)	**Potential range** δ (Å)	**potential wall** D_0 (Å)	**RMSE**
U_L+U_E	13.5±0.2	1.5±0.5	3.6±0.5	31.8±3.0	0.0073

Best fit	**2920** μM	**2300** μM	**1470** μM	**750** μM	**520** μM	**210** μM	**Weighted Average**
RMSE	0.0050	0.0049	0.0040	0.0048	0.0072	0.013	0.0073

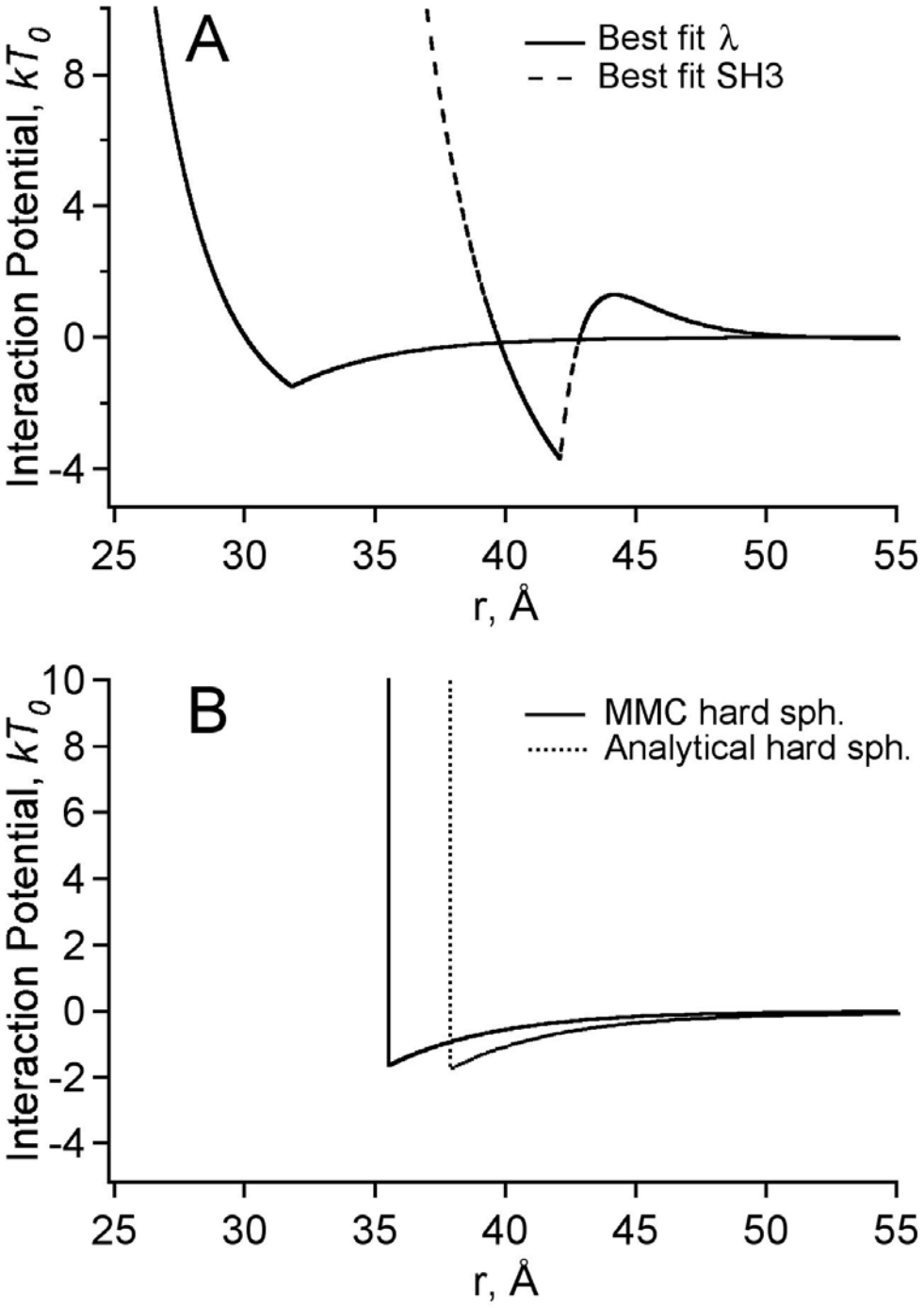


Figure 2.4 : (A) Best-fit interaction potential and (B) Comparison of the MMC and analytical best-fit

(A) Best-fit interaction potential for λ^*_{6-85} and *fyn*-SH3 (in 45% ethylene glycol buffer at –28 °C).

(B) Comparison of the MMC and analytical best-fit hard sphere + exponential potentials for λ^*_{6-85}. The greatest variation between the three λ^*_{6-85} shown is in D_0. (MMC parameters: D_0=35.5 Å, δ=4.14 Å, ε=1.65 kT_0, R_g = 13.8 Å; analytical: D_0=37.8 Å, δ=4.14 Å (fixed), ε=1.71 kT_0, R_g=13.6 Å).

2.4.3 LMD simulation for λ^*_{6-85}

We also performed a LMD simulation with the same potential as MMC at 2920 μM concentration, to confirm consistency of the MMC fitting results with molecular dynamics. We tested a range of different time scales (500 ns, 50 ns, 5 μs, and 20 μs) for 25 proteins in a cube having periodic boundary condition. The resulting $I(Q)$ is shown in [Figure 2.5A], and agrees with the experimental data within sampling uncertainty. The sampling uncertainty of the molecular dynamics simulations is shown by the error bars. The time scale between successive configurations chosen for scattering calculations was estimated form the diffusion equation $\langle r^2 \rangle = 6Dt$ in 3-D, allowing the protein ensemble to move enough so that successive configurations were independent of one another.

2.4.4 Concentration-dependent SAXS of *fyn*-SH3

[Figure 2.3B] shows the concentration dependence $I(Q)$ for *fyn*-SH3. The slope of a Guinier plot (not shown) deviates more strongly from linearity at low Q than for λ^*_{6-85}, indicating more extensive aggregation and a stronger interaction potential. As in the case of λ^*_{6-85}, the spherical monomer approximation works to the largest Q values for which data were collected.

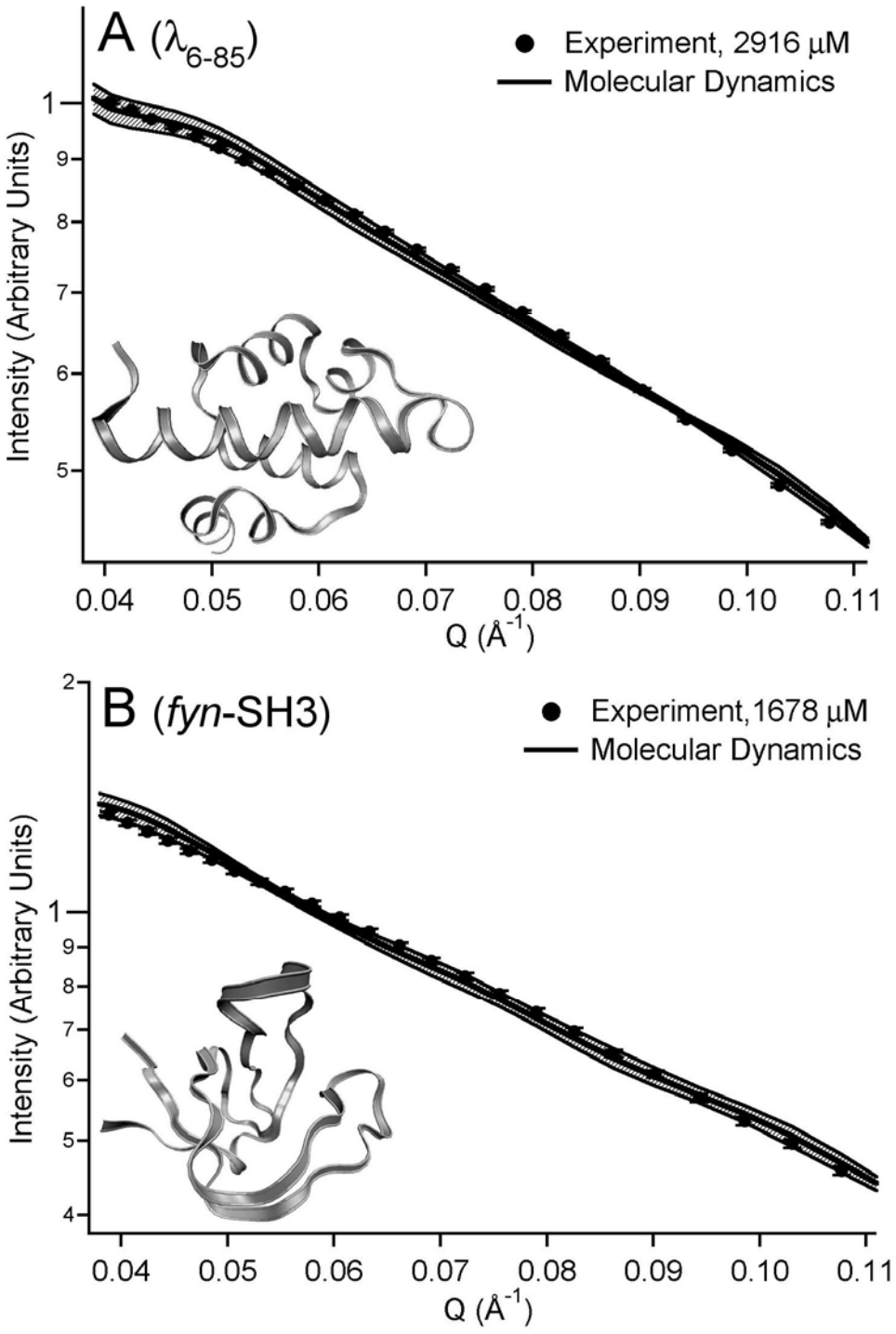


Figure 2.5 : SAXS data and MD simulation results

Experiment (circles with error bars) and molecular dynamics simulation (thick solid line) of the scattering intensity *vs*. magnitude of the scattering vector for λ^*_{6-85} (A), and *fyn*-SH3 (B), confirming the quality of the parameter set obtained by MC modeling. The estimated 1σ sampling error we achieved in the MD simulations is indicated by the envelopes. Native PDB structures for the protein fragments, as visualized with VMD [37], are shown as insets.

2.4.5 Fitting results for *fyn*-SH3

Ensemble configurations were generated by MMC simulation. The best fit [Figure 2.3B] was obtained with a $U_L + U_Y + U_E$ potential (Lennard Jones r^{12} repulsive wall + attractive Yukawa potential well + repulsive exponential potential).

$$U = \begin{cases} (\varepsilon_1 - \varepsilon_2)\left[\left(\frac{D_0}{r}\right)^{12} - 2\right] & (r < D_0) \\ -\varepsilon_1\left(\frac{D_0}{r}\right)\exp\left[-\left(\frac{r - D_0}{\delta_1}\right)\right] + \varepsilon_2 \exp\left[-\left(\frac{r - D_0}{\delta_2}\right)\right] & (r > D_0) \end{cases} \tag{2.7}$$

Potentials without a repulsive long range interaction produced significantly worse fits ($\chi^2/\chi^2_{\text{optimal}} > 2$). For the three-term potential, the calculated radius of gyration is 14.85 Å and the potential wall size is 42.0 Å. The attractive Yukawa potential depth is 11.2 kT_0 with a 1 Å range, The repulsive exponential potential depth is 7.5 kT_0 with a range of 2.0 Å, which results in a net potential depth of 3.65 k_BT ([Table 2.2] and [Figure 2.4A]). We used 625,000 Metropolis iterations to obtain converged results, and 50,000 final configurations were sampled. Compared to λ^*_{6-85}, SH3 consistently produced fits with shorter range, but deeper potential wells.

2.4.6 LMD simulation for *fyn*-SH3

We also performed a LMD simulation with the converged MMC potential at 1690 μM concentration, to confirm consistency of the MMC fitting results and the molecular dynamics simulations. Again, we tested a range of different time scales (from 50 ns to 5 μs) for 25 proteins in a cube having periodic boundary condition. The resulting $I(Q)$ is shown in [Figure 2.5B], and also agrees with the experimental data within sampling uncertainty. The time scale between successive configurations was chosen by the same criterion as for λ_{6-85}.

Table 2.2 : Best fit of the *fyn*-SH3 SAXS data to a $U_L + U_Y + U_E$ (r^{12} repulsive + Yukawa attractive + exponential repulsive) potential

Also shown are the root mean square errors (RMSE) for the best overall fit at individual concentrations. kT_0 corresponds to 245 K.

Potential type	R_g (Å)	Attractive		Repulsive		D_0 (Å)	Net well depth ε (kT_0)	RMSE
		ε_1 (kT_0)	δ_1 (Å)	ε_2 (kT_0)	δ_2 (Å)			
$U_L+U_Y+U_E$	14.85 ±0.2	11.2 ±0.5	1.0	7.5±0.2	2.0	42.0 ±4.0	3.7	0.035

Best fit	1680 μM	1020 μM	690 μM	470 μM	190 μM	60 μM	Weighted Average
RMSE	0.016	0.0074	0.010	0.034	0.028	0.060	0.035

2.5 Discussion

We have obtained interaction potentials for two proteins under identical buffer conditions by using the four-step procedure in [Figure 2.1]. First, Monte Carlo or molecular dynamics simulations of a model protein ensemble compute thermally averaged or time averaged particle distributions for up to 100 protein particles. Next, X-ray scattering functions $F_{total}(\mathbf{q})$ are computed directly for the whole ensemble. These are essentially exact for scattering angles corresponding to the size range from monomer particle to simulation box. In the third step, the resulting scattering intensity is computed without further approximations and then compared to SAXS data. In the last step, a least-squares algorithm refines the potential parameters, so that a new simulation can be started to iterate until the best fit is obtained. The best fits are summarized in [Table 2.1] and [Table 2.2].

Although MMC sampling and MD simulations are computationally much more expensive than the analytical approximations commonly used, direct simulation provides a correct description of the scattering amplitude at any concentration, for any monomer size, and for any aggregate shape consistent with the model monomers and up to the size of the simulation box. Any functional form of the potential, rather than a perturbing potential added to a hard sphere repulsion, can be fitted without additional effort simply by replacing the two-body interaction potential in the simulation.

The simplifying assumptions we retained in the present application are an isotropic interaction potential and hence an isotropic monomer shape, limiting the maximum Q values that could be fitted. The ratio $R_g / R_0 = 2R_g / D_0$ provides a connection between the interaction potential (characterized by D_0) and how the protein scatters (characterized by R_g). Both proteins had a ratio within 9% of the $\sqrt{3/5}$ ratio expected for spherical monomers ([Table 2.1] and [Table 2.2]). Over the Q-range we examined, neither deviations of protein shapes from a sphere nor electron density variations are likely to fully account for the difference from the ideal $\sqrt{3/5}$ ratio. More likely, hydration water that interacts strongly with the protein surface could explain the discrepancy between the fitted values of R_g and D_0 because the effective size of the hydrated protein could simply be different for the two different physical processes of X-ray scattering and protein-protein interaction.

Indeed, our fitted radii of gyration in [Table 2.1] and [Table 2.2] are larger than the values obtained by taking the bare protein structures from the Protein Data Bank. For example, one would expect R_g = 11.85 Å for bare λ^*_{6-85}, not the 13.1-13.8 Å range obtained from our fits, the best of which has R_g = 13.5 Å [Table 2.1]. It has been shown previously that the hydration layer around proteins perturbs SAXS such as to increase the effective radius by 1-2 Å. The program CRYSOL takes this effect into account [38, 39]. Its predicted hydrated radius of gyration is 13.5-13.8 Å depending on the method used, in excellent agreement with the value we derived from fitting interaction potentials to the SAXS experiment. A similar result is obtained for *fyn*-SH3, although our experimentally fitted radius of gyration is yet another 0.5 Å larger than the one obtained from CRYSOL. This could be due to the histidine tag on our *fyn*-SH3 protein, which was not included in

the CRYSOL calculation (no structure is available for the tag).

Extrapolations of the scattering data in [Figure 2.1] to zero concentration are fitted well by CRYSOL with Protein Data Bank structural data as input, showing that the folded monomer shapes remain consistent throughout the concentration range. Our fitting approach clearly does not require a low concentration extrapolation to yield reliable results.

This leads to the question: What range of concentrations is needed to reliably fit the potential parameters, and which parameters remain least reliably determined? The fitting uncertainties are largest for D_0. We confirm in two ways that D_0 is the least well constrained parameter in our fits of λ^*_{6-85} and *fyn*-SH3. First, we fixed it at the hard sphere value $2\sqrt{5/3}R_g$. This yielded radii of gyration R_g, well depths ε and potential ranges δ that agreed with [Table 2.1] and [Table 2.2] within the indicated uncertainties. D_0 on the other hand shifted by up to 11%, showing that R_g is much more strongly constrained by the SAXS data than is D_0. Still, the χ^2 of the fits did increase by up to 70% when the constraint relating D_0 and R_g was introduced. Thus the differences between D_0 and R_g cannot be explained just by parameter uncertainties.

To investigate how many concentrations are needed to determine parameters, we performed fits with as few as two of the concentration series. For example, 2920 and 520 μM for λ^*_{6-85} yielded a very similar potential shape (ε = -1.6 kT_0, δ = 3.8 Å for comparison with [Table 2.1]), but the parameter D_0 varies greatly (as low as D_0 = 25 Å). When more concentrations are added, D_0 approaches values more consistent with R_g. We conclude, at least for λ^*_{6-85} and *fyn*-SH3, that two concentrations are sufficient to define the shape of the potential, but that D_0 must either be constrained by R_g, or requires at least 5-6 concentrations, including high concentrations, to be adequately constrained.

It is worth noting that analytical fitting methods also have problems determining D_0 accurately. For example, two studies of the lysozyme interaction potential had to fix D_0 at values ranging from 28 to 36 Å in order to fit the other potential parameters [18, 40]. The value for an ideal sphere is about 37 Å in that case. Our numerical scattering method can be used to validate the analytical approximations usually used to obtain isotropic interaction potentials. To do so, we compared an analytical potential for λ^*_{6-85} to a

simulation-derived potential. To make the comparison feasible within the limitations of the analytical approach, we used a hard sphere reference potential, coupled with an attractive exponential term, to yield a potential similar in shape to our best fit in [Table 2.1]. We employed the analytical method described by Winter and coworkers [18], after verifying that our analytical code reproduced their experimental SAXS intensities from their potential parameters. [Figure 2.4B] compares the numerical λ^*_{6-85} potential with the analytical potential. Either D_0 or the potential range δ was highly correlated with potential depth in the analytical fit, so we had to fix one at the MMC value (δ in [Figure 2.4B]; the result looks even closer with D_0 fixed). With that restriction, reasonable agreement is obtained between the analytical and simulation result. However, as already discussed above, the simulation yields a much more robust fit than the analytical model when more than 2 concentrations are used; it does not treat the potential as a small perturbation to a hard-sphere wall. In particular, D_0 can be floated as a free parameter and yields results consistent with R_g (<9% discrepancy) when enough concentrations are fitted. To the best of our knowledge, we did not find any analytical treatments in the literature where adjusting D_0 and R_g independently was possible, let alone yielded consistent results.

We examined a number of isotropic interaction potentials in addition to the best-fit and hard-wall shapes, and found that Gaussian attractive potentials generally performed more poorly than the exponential or Yukawa forms used in the DLVO model. In all fits, the λ^*_{6-85} potential was longer range than the *fyn*-SH3, which resembles a 'sticky sphere.' A long range but weak attractive potential for λ^*_{6-85} is compatible with recent terahertz measurements of hydration shells around the same mutant [19]. These measurements indicated that the dynamics of water molecules are affected by the protein to > 10 Å from the protein surface. Such hydration water may significantly mediate protein-protein interactions. It is even possible that the protein-protein interaction potential depends on protein-concentration because of concentration-induced changes in the hydration shell. However, our current SAXS data was adequately modeled by a concentration-independent interaction potential.

λ^*_{6-85} has a significantly lower propensity for aggregation than *fyn*-SH3, but

only the latter requires a repulsive potential in the fit to match the data within experimental uncertainty [Figure 2.4]. Both proteins were examined in identical buffer solutions of 45%/55% by volume ethylene glycol/water, 50 mM phosphate at pH 7.0 and –28 °C. As discussed by Winter and coworkers [18], the size of the repulsive potential is very sensitive to the ionic strength and ionic composition of the buffer. Given the isoelectric points of pI = 8.25 (λ^*_{6-85}) and pI =4.84 (*fyn*-SH3), it is not surprising that there are differences between λ^*_{6-85} and *fyn*-SH3 in the screening of the long-range electrostatic repulsion.

As measurements over wide Q-ranges become available with new high brightness synchrotron sources, the direct fitting approach will also be useful for determining anisotropic interaction potentials. This requires two additions to our treatment: the potential itself must treat anisotropic interactions, and the scattering calculation can no longer assume spherical monomers. Regarding the potential, Ha-Duong and coworkers have developed residue-residue pair potentials that can be applied to surface residues of interacting proteins [41]. To treat arbitrary protein shapes one adds a rotational diffusion term to [Equation (2.3)], and replaces F_m in [Equation (2.4)] by the orientation-dependent structure factor of the monomeric protein computed with a program such as CRYSOL [38]. It remains to be seen how much information might be extracted from scattering data at larger angles using this approach.

In conclusion, direct fitting of SAXS data to interaction potentials via Monte Carlo or molecular dynamics simulation of a model protein ensemble provides a useful alternative to analytical approximations. The form of the potential is unrestricted and no approximations regarding the scattering amplitude of the model protein ensemble need to be made. A range of concentrations still provides the best sampling of protein-protein distances to determine the potential (the potential wall location D_0 in particular), but extrapolations to zero concentration are not necessary. When the potential is restricted to have a hard sphere wall, our method validates the analytical methods used to date, but actually fits D_0 more consistently with the protein size determined by the scattering amplitude (R_g). With the advent of higher power computing, the numerical approach demonstrated here can be extended straightforwardly to include coarse-grained anisotropic interaction potentials, and randomly reorienting non-spherical protein shapes.

2.6 Acknowledgments

We thank Dr. Liang Guo at Argonne National Laboratory for helpful assistance in setting up the beam line for experiments, and Mr. Y. Matsumura for assistance during data collection.

This work was supported by National Science Foundation grant MCB 0613643. Time on BioCAT-18, managed by Prof. Thomas Irving, was supported by proposal GUP-6360 of the Advanced Photon Source at Argonne National Laboratory. M.G. was supported by a Lycan Professorship.

Chapter 3 The Terahertz dance of water with the proteins: The extended dynamical hydration shell probed by Terahertz spectroscopy

3.1 Abstract

The focus in protein folding has been very much on the protein backbone and side chains. However, hydration waters make comparable contributions to the structure and energy of proteins. The coupling between fast hydration dynamics and protein dynamics is considered to play an important role in protein folding.

We show here that Terahertz spectroscopy directly probes such hydration dynamics around proteins, and determines the width of the dynamical hydration shell. We observe an unexpected non-monotonic trend in the measured terahertz absorbance of the lambda repressor fragment as a function of concentration. The trend can be explained by overlapping hydration layers around the proteins. The experimental data suggest an influence on the correlated water network motion beyond 20 Å, greater than the pure structural correlation length usually observed so far.

We also use terahertz (far-infrared) spectroscopy to probe directly the effect of mutations and solvent pH on the hydration shell-protein interaction. We find that the pseudo-wild-type has a much more pronounced effect on long distance hydration water than mutants that have decreased helix stability. Disturbing the pseudo-wild-type at pH 2 likewise reduces the long distance hydration effect, which indicates the hydrophobicity significantly affects hydration water structure.

3.2 Introduction

Hydration water plays an integral role in the folding and function of proteins. For

example, the expulsion of hydration water sheets from the hydrophobic core has been implicated as a major cause of the final folding barrier leading up to the native state.[42-44] Specific functional water molecules have been resolved by NMR spectroscopy and X-ray crystallography, for example mediating water transport through pores.[45, 46]

Water molecules interact (or are highly correlated) with proteins on many length and time scales. Although the dynamics of the hydration water occurs on the picosecond time scale, ‘‘slaving’’ [47] to fast solvent modes profoundly affects the slower but larger-scale protein motions [48]. In return the protein influences the structure and dynamics of surrounding water molecules.[49] X-ray crystallography has revealed ordered water structure around polar and charged side chains [50], as well as cooperative insertion of water into hydrophobic cavities.[51] Dielectric spectroscopy extends the time scale from microseconds down to 0.1 ns.[52] Experiments have been extended to the Terahertz range in films and crystals, probing motions on the picosecond time scale.[53, 54] Hydrated protein powders probed by inelastic neutron scattering (0.1–100ps) or solid-state NMR (nanoseconds) reveal that slower protein time scales and faster solvent time scales indeed show correlated dynamics.[55] On the fastest time scales, 2D infrared spectroscopy and fluorescence of surface residues provide local probes of the dynamics in the femtosecond to picosecond range.[56, 57]

Terahertz absorption spectroscopy of biomolecules fully solvated in water yields direct information on the global dynamical correlations among solvent water molecules. And the Terahertz absorption coefficient is even more sensitive to fast water dynamics than dielectric spectroscopy or IR spectral changes.[58] Yet, Terahertz spectroscopy is experimentally challenging,[59] because of the strong Terahertz absorption of water.

The Havenith group at Ruhr-University-Bochum, Germany has devised table-top Terahertz sources capable of penetrating the bulk of aqueous solutions.[60] With the advent of powerful table-top Terahertz sources, a new window between microwaves and the infrared is opening up onto the interaction of water molecules with proteins. Even more, THz radiation is safe for biological samples because it is non-ionizing, unlike X-rays. Terahertz spectroscopy has been demonstrated as a new probe of the coupling between biomolecules and their hydration shells [19, 61-63], because key large

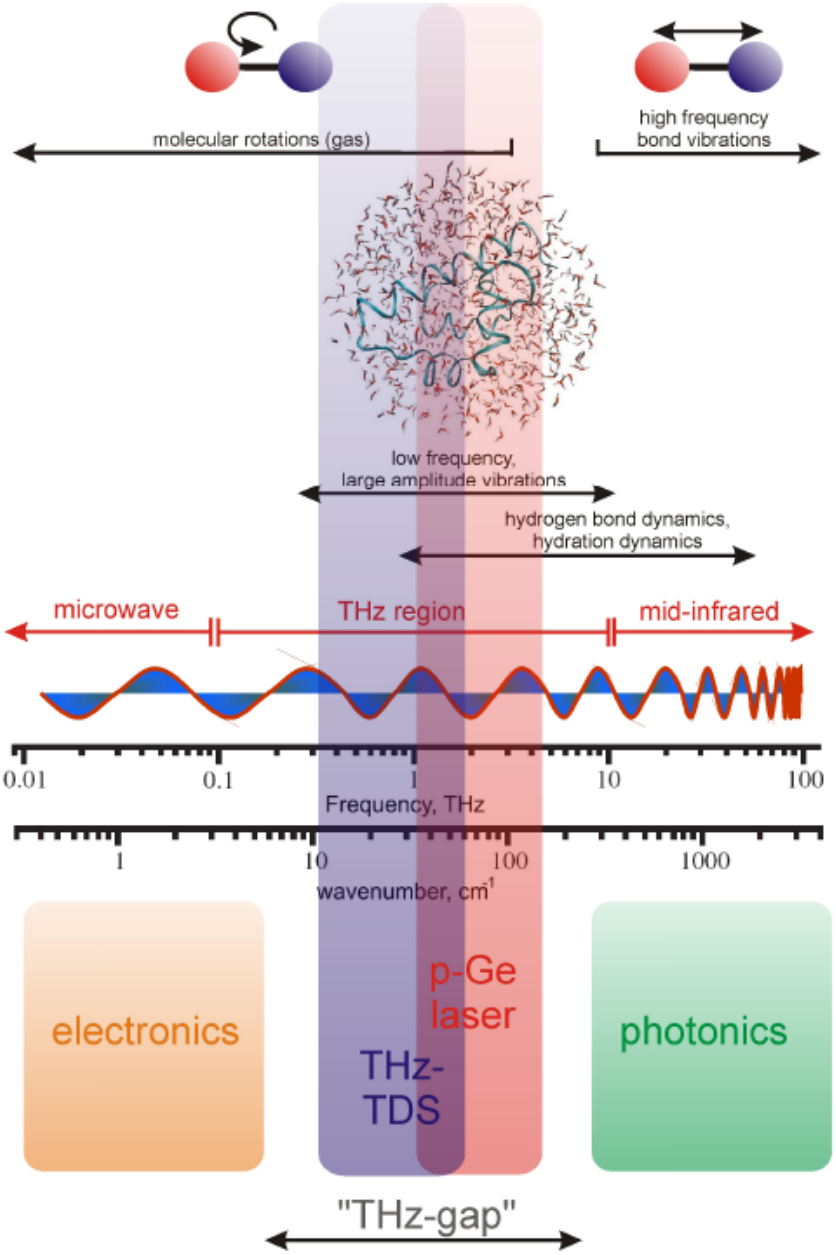


Figure 3.1 : The electromagnetic spectrum and typical resonant molecular transitions[2]

The Terahertz region is located between microwaves and mid-infrared. Molecular transitions within this region were difficult to probe, the region has therefore been termed the "Terahertz-gap". The experimental setups, the p-germanium laser (p-Ge laser, **red**) and the Terahertz time domain spectrometer (Terahertz-TDS, **blue**), are developed by the Havenith group at Ruhr-University-Bochum, Germany. They are used to probe rotational transitions and hydration dynamics within this frequency region. The p-Ge laser is tunable from 1 Terahertz to 4.5 Terahertz and the Terahertz-TDS is suitable for Terahertz spectroscopy in the region from 0.1 Terahertz to 2 Terahertz. From ref.[64]: Ebbinghaus, S., *THz Spectroscopy of Biomolecules*, in *Ph.D. Thesis in Chemistry and Biochemistry*. 2007, Ruhr-University-Bochum: Bochum, Germany.

[2] By courtesy of Simon Ebbinghaus, Ruhr-University-Bochum, Germany

amplitude motions of water and biomolecules occur on the picosecond timescale – the typical characteristic time of Terahertz spectroscopy. [Figure 3.1] briefly describes the electromagnetic spectrum and typical resonant molecular transitions monitored by Terahertz spectroscopy.

Terahertz frequency range probes the intermolecular collective modes of the hydrogen bonding network and some collective modes of the protein, such as skeletal and breathing modes. Using a free electron laser, Plaxco, Allen and co-workers showed that terahertz absorption decreases linearly when large concentrations of protein are added to the solution.[62] Such behavior indicates that the solute molecules replacing the water have a lower absorption within this frequency range. A coupling of Terahertz hydration dynamics and protein dynamics was also suggested by spectroscopy of hydrated bacteriorhodopisin films.[65] We use Terahertz spectroscopy of lambda repressor fragments and ubiquitins to study the correlation between protein structural flexibility and the absorption properties of the extended dynamical hydration shell around the protein. Such studies can now be carried out systematically in the laboratory thanks to the advent of table-top Terahertz radiation sources with sufficient output power to penetrate aqueous solutions of proteins.[60] To tune the protein flexibility, we various mutations known to decrease structural rigidity of the protein, as probed by fluorescence anisotropy.[66]

3.3 Materials and methods

3.3.1 Lambda repressor mutants

The lambda repressor fragment 6-85 Tyr22Trp mutant gene, a gift from Terry Oas, was expressed in Escherichia coli BL-21 cells and purified, as described in ref. [20]. (Also see Chapter 2.3.1 and Appendix B) The resulting λ^*_{6-85} protein was buffered in 50 mM magnesium acetate (pH 7.3) at the concentrations of up to 2.3 mM, where data

could be taken without signs of precipitation in the 15–22°C range. Small-angle x-ray scattering data have shown that the protein does not cluster up to twice this concentration in aqueous ethylene glycol solvents, as described in ref [13].

The protein we used in Terahertz spectroscopy contained the mutations ([Tyr22Trp, Glu33Tyr, Gly46Ala, and Gly48Ala] and [Tyr22Trp, Glu33Tyr, Ala37Gly, and Ala49Gly]) as shown in [Figure 3.2]. They are engineered by site-directed mutagenesis (Stratagene Quickchange kit, La Jolla, CA) based on a wild-type plasmid donated by Terry Oas [25].

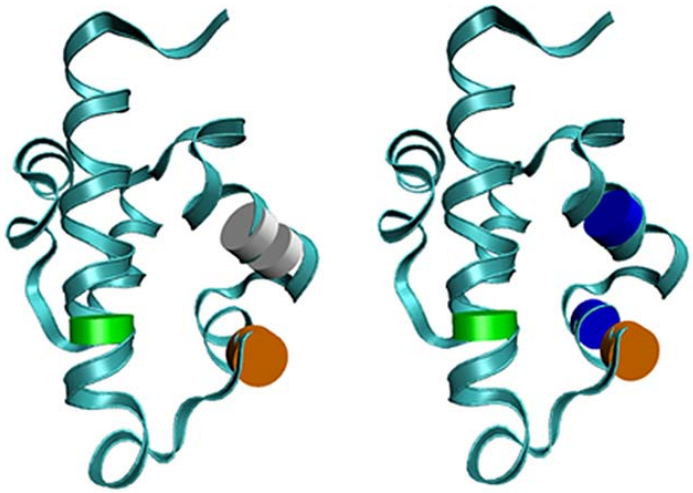


Figure 3.2 : Lambda repressor mutants

Left: Mutants of Tyr22Trp (green), Glu33Tyr (brown), Gly46Ala (gray1), and Gly48Ala (gray2)
Right: Mutants of Tyr22Trp (green), Glu33Tyr (brown), Ala37Gly (blue1), and Ala49Gly (blue2)
These pictures are generated by VMD to show its 3D shapes.[37]

λ^*_{6-85} displaces 16,000 Å^3 of buffer, based on the hydration-free radius of gyration (R_g) of 12.1 Å, which is estimated from small angle x-ray scattering data by Dumont *et al*.[13]. Assuming a homogenous hard sphere, one can calculate a surface radius ($R_{surface}$) [67] as

$$R_{surface} = \sqrt{\frac{5}{3}} R_g \approx 15.6 \text{ Å} \tag{3.1}$$

3.3.2 Ubiquitin mutants

Ubiquitin is a small α/β protein with one α-helical segment and a short 3_{10} helical segment.[68] We studied the wild-type human sequence, obtained from Sigma. In addition, we introduced a Phe45Trp mutation to have a strongly fluorescent residue as an independent probe of protein flexibility.[69] We call this pseudo-wild-type Ub*, and the wild-type Ub.

Each ubiquitin molecule, with a bare radius of gyration of R_g = 11.7 Å estimated from the X-ray crystal structure and different MD simulations,[70] has a surface radius of $\sqrt{5/3}R_g \approx$ 15.1 Å and displaces ca. 14400 Å^3 of buffer.

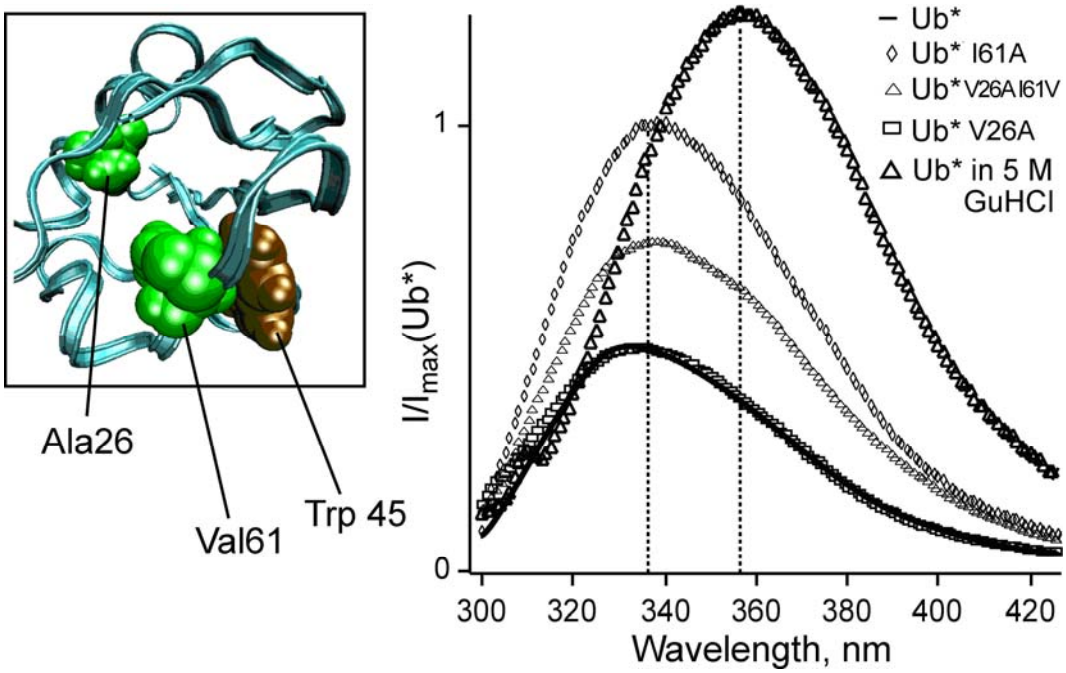


Figure 3.3 : Ubiquitin structure and fluorescence profiles of its mutants

Left: structural model of Ub* V26A I61V, obtained by making side chain substitutions with XPLOR, using the SOLVATE feature of VMD to add TIP3P water, then relaxing the structure at 298 K for 8 ps in an NVT ensemble. **Right**: fluorescence intensities of mutants compared to Ub* and denatured Ub*. Note that despite different fluorescence intensity maxima, all mutants have peak shifts close to native Ub*. Data are from refs. [66, 71]

[Figure 3.3] shows the two sites we chose for single- and double-point mutations to increase protein flexibility. Ile61 is in van der Waals contact with Trp45, and Val26 has at least one intervening residue to Trp45.[66] The three mutants of Ub* represent

two single and one double truncation and are summarized in [Table 3.1]. All mutations truncated aliphatic side chain residues to shorter aliphatic side chains, to avoid direct effects on the tryptophan fluorescence quenching via introduction or elimination of electron transfer, proton transfer, or Förster mechanisms. We use the abbreviation Ub*X##Y, where * indicates the tryptophan, X is the original amino acid before mutation, in position of residue number ##, and Y is the amino acid after mutation.

Table 3.1 : Parameters for the ubiquitin mutants

Mutant	Abbreviation	ΔG, kJ/mole	f_{ratio}
Wild-type	Ub		1.0
Pseudo-WT (F45W)	Ub*	-34±1.5	1.0
F45W / Ile61Ala	Ub* I61A	-18±1	2.0
F45W / Val26Ala	Ub* V26A	-20±1.5	1.0
F45W / Val26Ala / Ile61Val	Ub* V26A I61V	-14±1	1.4

ΔG is the folding free energy at 0 M GuHCl, 25 °C, pH 5.9 in 40 mM phosphate buffer from ref. [66]. f_{ratio} is the peak ratio of fluorescence intensity compared to Ub*.

The plasmids for Ub* I61A, Ub* V26A and Ub* V26A I61V were obtained by single point mutations of the original Ub* plasmid (provided by Tracy Handel) using site-directed mutagenesis (Stratagene). Proteins were over-expressed in *E. coli* (BL21) and purified as indicated elsewhere.[72] Purity was checked by gel electrophoresis (SDS-PAGE) and protein identity by low resolution mass spectroscopy. Samples were lyophilized and kept at -20 °C before use.

Ubiquitin mutants were re-suspended in buffer for terahertz (Terahertz) absorption studies at (20 ±0.5) °C. Unless otherwise indicated, all solvents were buffered with 50 mM magnesium acetate buffered at pH 4.8. Protein concentration was varied between 0 and 3.6 mM. The actual concentration was measured with an uncertainty of 3% using UV absorption at 280 nm, and assuming an extinction coefficient of 6970 $M^{-1}cm^{-1}$.[27] We

did not observe any aggregation below a concentration of 3.8 mM.

[Figure 3.3] summarizes the effect of the side chain truncations studied here on protein flexibility, relative to Ub*. As shown in reference,[66] local flexibility of the ubiquitin structure is directly correlated with tryptophan fluorescence intensity. The Ub* I61A and Ub* V26A I61V mutants with a truncation adjacent to Trp45 have greatly increased fluorescence intensity compared to Ub*. The truncation mutant Ub* V26A looks very similar to Ub* because tryptophan is not in contact with residue 26 and probes flexibility only locally, but molecular dynamics simulations showed that the Val26Ala core truncation induces a local increase in flexibility greater than the Ile61Ala near-surface truncation. Such truncations typically decreased the anisotropy parameter $< \cos^2 \theta >$ of the tryptophan side chain from 0.95 to 0.85, with excursions as low as 0.55 (isotropic: 0.5) for the Ub* V26A I61V double mutant.

3.3.3 Terahertz p-type Germanium laser spectrometer

Using a novel Terahertz p-type Germanium laser spectrometer [60] built by the Havenith group (Ruhr-University-Bochum, Germany), we have measured the change of the absorption coefficient of the proteins in the spectral range from 2.1-2.8 Terahertz. The Havenith group has built two different configurations for a Terahertz p-type Germanium laser spectrometer. The first one is a single beam configuration at a varying layer thickness [Figure 3.4, on the right], and it was used for the measurement of an extended dynamical hydration shells [see Chapter 3.4]. The other improved configuration is a double beam configuration at a fixed layer thickness [Figure 3.4, on the left], and it was used for the measurements in [Chapter 3.5] and [Chapter 3.6].

By using both configurations, we determined the protein absorption relative to the buffer, showing a non-linear behaviour of the integrated Terahertz absorption with increasing protein concentrations for all samples.

***Double beam configuration (at a fixed layer thickness)**:*

To measure the difference in absorption between protein solution and buffer blanks accurately, we set up a Terahertz difference spectrometer [Figure 3.4, on the left]. This approach also minimized any additional systematic errors due to temperature drifts or changes in the air humidity as would be present in case of subsequent measurements. Specifically, we determine:

$$\alpha(c) = \alpha_{probe}(c) - \alpha_{buffer}(c) \tag{3.2}$$

with $\alpha_{probe}(c)$ and $\alpha_{buffer}(c)$ being the integrated absorption coefficients (2.1-2.8 Terahertz) of the probe and buffer at a given concentration c.

Using the double beam configuration, the pulse train is splitted by a chopper (5) into one part probing the sample absorption and a second part probing the reference absorption. Both beams are recombined by a second chopper (7) and detected. The transmitted intensities were measured at a fixed layer thickness using a standard Bruker liquid sample cell with teflon spacers and z-cut quartz windows. The layer thickness of the aqueous sample was determined to be $(52.6 \pm 0.3)\,\mu m$ using FTIR-spectroscopy. The temperature of the sample was kept at (20±0.5) °C by using a Peltier element. The measured humidity near the purged sample cell was below 8%.

Each signal was detected by a gated integrator. In order to further minimize systematic errors, we interchanged the sample and reference channel at each concentration. Measurements were repeated five times at each concentration to provide an error estimate for the absorbance difference. Each point corresponds then to the average of 10,000 pulses. The main error source was found to be the manual refilling of the sample cell, which leads to slight sample-to-sample, cell positioning, or pathlength variations.

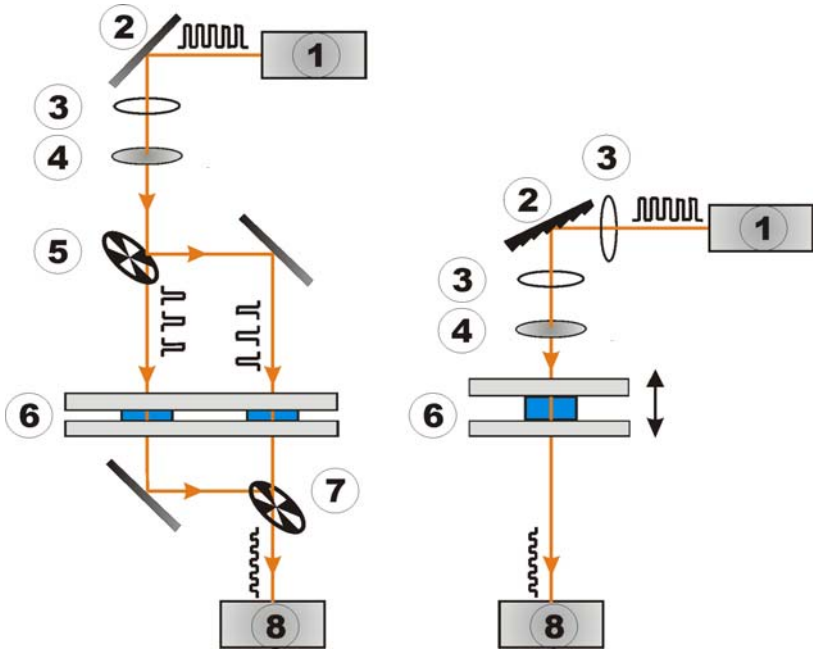


Figure 3.4 : Terahertz p-type Germanium laser spectrometer[3]

Two configurations of the transmission spectrometer are shown. A double beam configuration (**left**) using a sample cell (6) with constant sample layer thickness and a single beam experiment using a sample cell with variable sample thickness (**right**). The elements of the spectrometers are: p-Ge laser (1), mirror or blazed grating (2), pinhole (3), polyethylene lens (4), reflecting chopper (5, 7), detector (8). For the single beam experiment, the complete pulse train emitted from the p-Ge laser (illustrated in black, not to scale) is transmitted through the sample. Using the double beam configuration, the pulse train is splitted by a chopper (5) into one part probing the sample absorption and a second part probing the reference absorption. Both beams are recombined by a second chopper (7) and detected. From ref. [64]: Ebbinghaus, S., *THz Spectroscopy of Biomolecules*, in *Ph.D. Thesis in Chemistry and Biochemistry*. 2007, Ruhr-University-Bochum: Bochum, Germany.

By data evaluation with Beer's law the absorption coefficient of the sample and the reference in the two separate channels were determined.

$$I(v,d) = I(v,0)\exp\left(-\alpha(v)d\right) \tag{3.3}$$

where α is an absorption coefficient, v is a Terahertz frequency, and d is a layer thickness.

Single beam configuration (a varying layer thickness):

[3] By courtesy of Simon Ebbinghaus, Ruhr-University-Bochum, Germany

The experimental setup is described in [Figure 3.4, on the right]. [64] For the single beam experiment, the complete pulse train emitted from the p-Ge laser is transmitted through the sample. The frequency separation is achieved by a blazed aluminum grating (2). The sample solution is sealed in a polyethylene (PE) bag and placed in the sample chamber with a variable thickness.

The transmitted intensity was measured as a function of the layer thickness d (which was varied in steps of 5 μm). The Terahertz absorption coefficient is obtained by scanning a variable-pathlength (d) cell and fitting the transmitted Terahertz power I according to Beer's law [Equation (3.3)], after subtracting a constant baseline.

3.4 An extended dynamical hydration shell around λ^*_{6-85}

3.4.1 Two component excluded volume model

If the proteins were completely transparent or much less absorbing ($\frac{\alpha_{\text{Protein}}}{\alpha_{\text{Buffer}}} \ll 1$) at 1.1-2.8 Terahertz, we would expect a linear decrease of the Terahertz absorption coefficient α with increasing protein concentration c_{Protein} in a solvent volume V [Figure 3.5]. The two component excluded volume model would fit the absorption coefficient α as a function of the concentration, c_{Protein}.

$$\begin{aligned}
\alpha &= \alpha_{\text{Protein}} \frac{V_{\text{Protein}}}{V} + \alpha_{\text{Buffer}} \frac{V - V_{\text{Protein}}}{V} \\
&= \alpha_{\text{Buffer}} - \frac{V_{\text{Protein}}}{V}(\alpha_{\text{Buffer}} - \alpha_{\text{Protein}}) \\
&= \alpha_{\text{Buffer}} - \frac{c_{\text{Protein}}}{\rho_{\text{Protein}}}(\alpha_{\text{Buffer}} - \alpha_{\text{Protein}}) \\
&\approx \alpha_{\text{Buffer}} \left(1 - \frac{c_{\text{Protein}}}{\rho_{\text{Protein}}}\right)
\end{aligned} \tag{3.4}$$

In this equation, the protein has concentration c_{Protein} in total solution volume V and ρ_{Protein} is the protein density of ca. 1.4~1.5 g/cm^3. [73] The approximation in the fourth line ($\frac{\alpha_{\text{Protein}}}{\alpha_{\text{Buffer}}} \ll 1$, such as plotted in [Figure 3.5], corresponds to the limit where protein absorption is negligible compared to the buffer absorption.

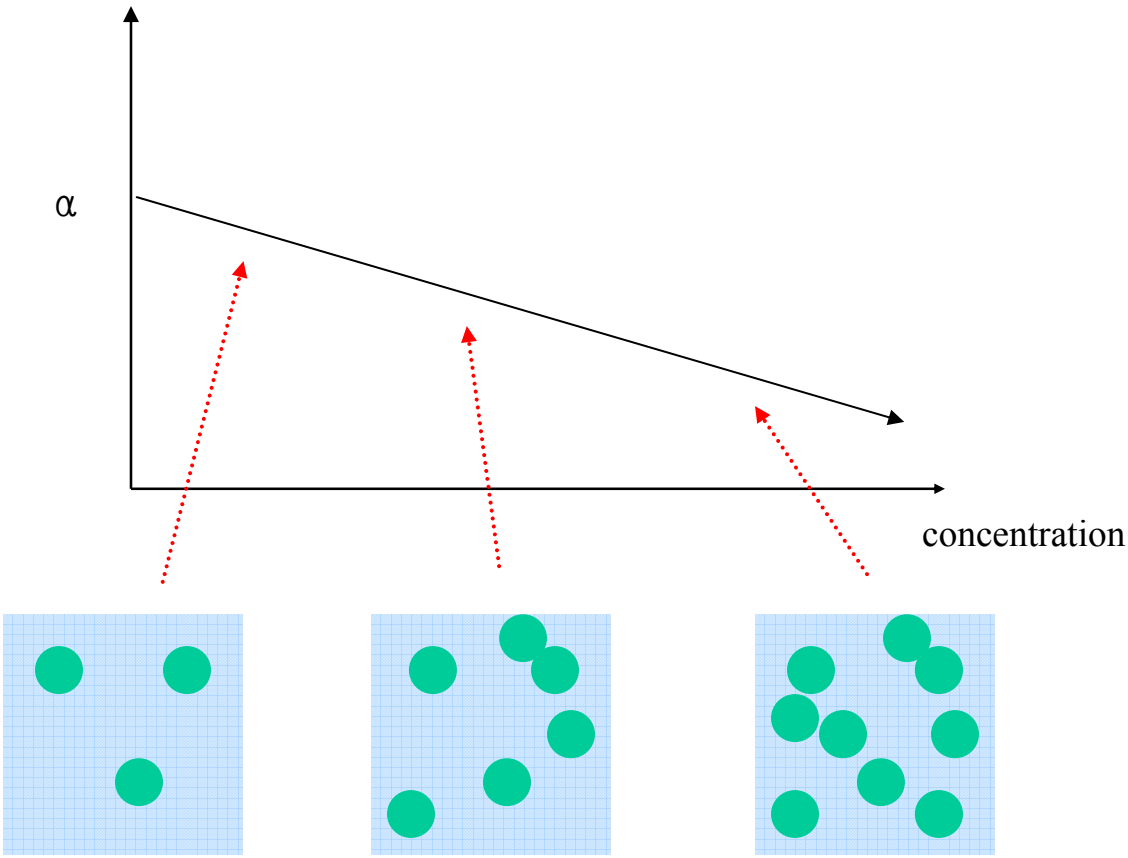


Figure 3.5 : Two component excluded volume model

Terahertz absorption decreases linearly as transparent or much less absorbing proteins replace the bulk water.

Any two-component model which considers only the absorption of the buffer and the ubiquitin wild-type would lead to the linear concentration dependence, although the slope would differ from [Equation (3.4)] in a more sophisticated dielectric cavity model.

3.4.2 Non-linear concentration dependence of λ^*_{6-85}

[Figure 3.6] displays the absorption coefficient relative to bulk water as a function of the concentration of λ^*_{6-85} at 2.25 Terahertz[4]. The proteins were dissolved in magnesium acetate buffer at pH 7.32.

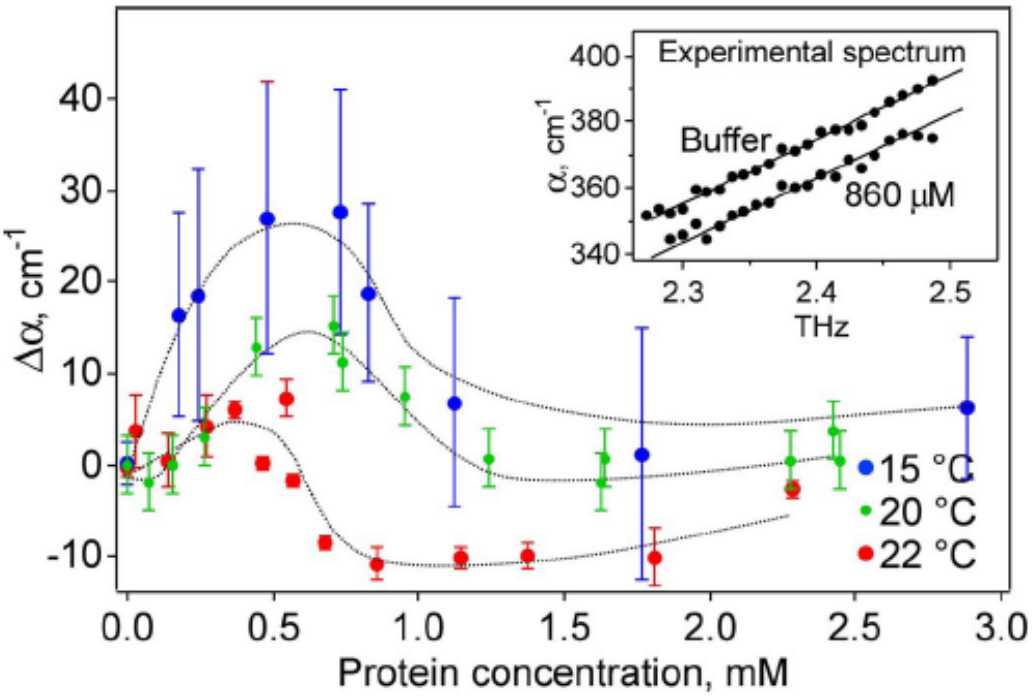


Figure 3.6 : Difference in the Terahertz absorption coefficient relative to bulk water

Plotted against concentration to 3 mM at 15°C, 20°C, and 22°C (more extensive averaging was done at 22°C because of the slightly smaller effect) in pH 7.3. The absorbance depends nonlinearly on concentration in this region. Note that the Terahertz absorption for bulk water (zero point) increases with increasing temperature. (**Inset**) The frequency dependence of the absorption coefficient is linear between 2.25 and 2.55 Terahertz (22°C: comparison of buffer and at a protein concentration of 860 µM).

The absorption coefficient increases before dropping, leading to the non-monotonic, non-linear concentration dependence. The 0.5–1.0 mM concentration at the turnaround in [Figure 3.6] corresponds to a water volume decrease of 1%. The measured Terahertz absorption deviates strongly from a linear decrease as predicted according to [Equation (3.4)]. Although at higher concentrations it will decrease quasi-linearly as

[4] All protein samples are carefully prepared and provided by the author, and the Terahertz data were collected by Simon Ebbinghaus in the Havenith group, at Ruhr-University-Bochum, Germany.

discussed in [Chapter 3.4.1] and in ref. [62], such a non-monotonic, nonlinear behavior observed in [Figure 3.6] cannot be explained just by a two component excluded volume model.

We have measured the concentration dependence of the Terahertz absorption at three different temperatures. We have a less error bar at higher temperature, since the absolute overall Terahertz absorption increases. [58] Although the absolute differences relative to the bulk value differ for the three temperatures, the overall variation in the absorbance with concentration is the same at each temperature. When we compare the three curves one has to keep in mind that the zero point (the bulk water value at the given temperature) decreases with decreasing temperature. [58] This partially explains the offset between the three curves. Whereas the absolute Terahertz absorption coefficient of water (c=0) is increased by approximately a factor of two for a temperature increase of 20°C at 2.0 Terahertz [58], a less pronounced change of the Terahertz absorption of the protein is expected. In this case $\Delta\alpha = \alpha(c) - \alpha(0)$, where $\alpha(c)$, the Terahertz absorption coefficient for a given concentration c, is expected to deviate at higher concentrations for different temperatures. The offset reflects the difference between the decrease for bulk water and protein + hydration water.

In addition, within our measurement uncertainty, the absorption of the solvated protein increased linearly with frequency in this rather narrow frequency range (see Inset in [Figure 3.6]). Therefore we used a linear fit in the measured frequency to obtain accurate absorption coefficients at a given frequency. This procedure, together with averaging over multiple measurements, minimizes noise and allows a reliable comparison between the different Terahertz absorption spectra for different protein concentrations.

3.4.3 Three component excluded volume model: *Explanation of the nonlinearity*

The minimum fitting model required to even qualitatively explain this deviation must incorporate at least a third component, attributed to water in the dynamical hydration shell around the protein, whose absorption coefficient is increased compared to

bulk water by the presence of the protein:

$$\alpha = \alpha_{\text{Protein}} \frac{V_{\text{Protein}}}{V} + \alpha_{\text{Shell}} \frac{V_{\text{Shell}}}{V} + \alpha_{\text{Buffer}} \frac{V - V_{\text{Protein}} - V_{\text{Shell}}}{V} \tag{3.5}$$

Thus, the hydration water around the protein must contribute in a nontrivial way to the total Terahertz absorption [Figure 3.7].

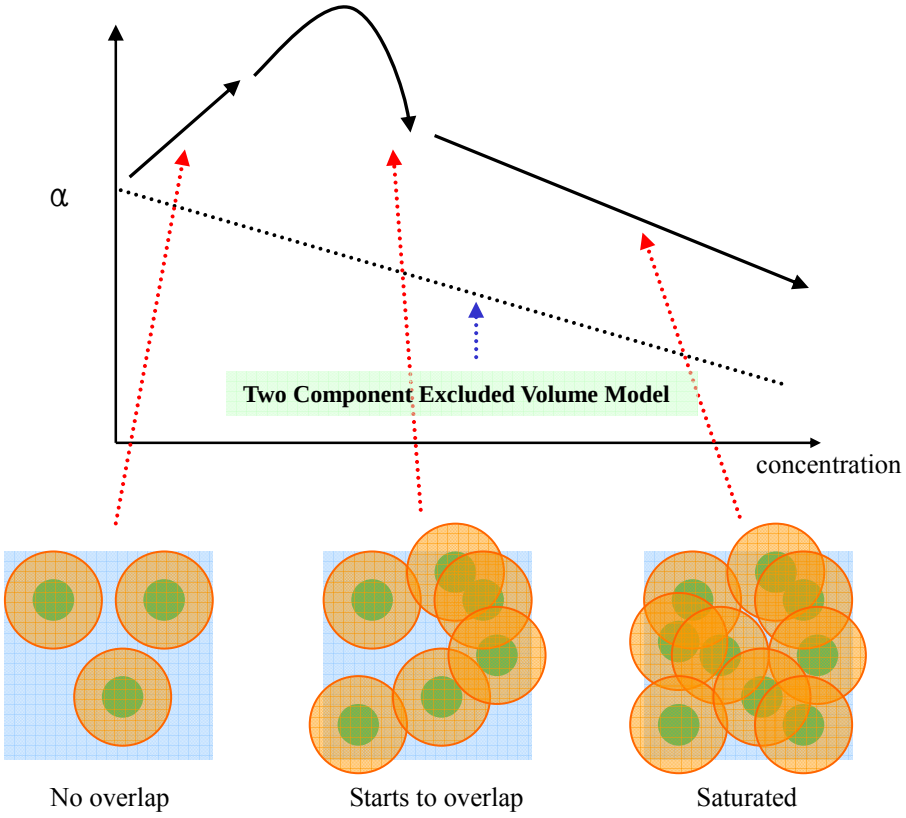


Figure 3.7 : Three component excluded volume model

The absorbance of the hydration shells depends on the distance between protein molecules. In general, the hydration shells absorb more than bulk water in Terahertz frequency.

Note that in this model, the volume of the hydration shell increases linearly with protein concentration at low concentrations. If the absorbance of the dynamical hydration shell exceeds the absorbance of the bulk water displaced by the shell and protein, the overall absorption will at first increase linearly with protein concentration. Eventually, the dynamical hydration shells overlap and get saturated, and their volume

actually decreases relative to the increasing volume of protein. As a result, there is a turnover (deviation from the linearity) in the absorption coefficient.

In the extreme limit of hexagonal packing of proteins and negligible protein absorption compared to the solvent, $\alpha \rightarrow 0.29\alpha_{\text{Shell}}$ if the shell is wide enough to displace all bulk water in the interstitial spaces. Thus unless the shell absorption coefficient exceeds bulk water by a factor of at least three, absorption will drop below the bulk value if we assume α_{Protein} is much smaller than α_{Buffer}.

3.4.4 A dynamical hydration shell extends to more than 20 Å

According to the three component excluded volume model, we can expect a turnover at the concentration where the hydration shell starts to overlap. [Figure 3.6] indicates that the cross over to the plateau is observed at the concentration of less than 1.0 mM, for all three temperatures. At the concentration of 1.0 mM, an average protein - protein center of mass distance D of ≈ 73.5 Å.

$$\frac{4\pi}{3}D^3 \simeq \frac{10^{-3}m^3}{0.001\times(6.02\times10^{23})} \tag{3.6}$$

If we take a surface radius of 15.6 Å for the bare ubiquitin [Equation (3.1)], the average distance between the protein surfaces at 1.0 mM is ~21.2 Å.

$$\begin{aligned} D &= 73.5\,\text{Å} \\ &= 2\text{R}_{\text{shell}}\left(\sim 21.2\ \text{Å}\right) + 2\text{R}_{\text{surface}}\left(\sim 15.6\ \text{Å}\right) \end{aligned} \tag{3.7}$$

We can then directly deduce the average size of the hydration shell, which corresponds to R_{shell} at the concentration at the point of turnaround in the Terahertz absorption to be ~21.2 Å. This corresponds to ca. 6 hydration water shells (ca. 3 Å average extension per water molecule), a significant range beyond just hydration waters interacting directly with the protein surface, and similar to values that have been reported

for carbohydrates by Terahertz spectroscopy.[74] Such long range interactions imply that cytoplasmic water, at concentrations of protein, RNA and carbohydrates in the 300 mg/ml range, is mostly 'biological water,' and not bulk water, at least by the Terahertz criterion.

3.4.5 Supported by MD simulation: The Terahertz absorbance of the hydration shell depends on the distance between proteins

However, even this three component model is unable to describe accurately the experimentally observed concentration dependence in the Terahertz absorption coefficient, unless the absorbance of the hydration water depends on the distance between protein molecules. In order to come to a microscopic understanding of the observed results, our collaborators, M. Heyden, X. Yu, and D. Leitner have carried out accompanying molecular modeling calculations, which reveal and quantify the protein distance dependence of the absorbance of the hydration shell. [19]

I'll present their methods and results briefly here. In molecular modeling calculations, the absorption coefficient $\alpha(v)$ is computed from the dipole autocorrelation obtained from MD simulations as

$$\alpha(v) = \frac{16\pi^4 v\left[1-e^{-hv/k_BT}\right]}{3hcn(v)} I(v) \tag{3.8}$$

where

$$I(v) = \tfrac{1}{2\pi}\int_{-\infty}^{\infty} dt\, e^{-i2\pi vt} < \overrightarrow{M}(0)\cdot\overrightarrow{M}(t) > \tag{3.9}$$

$\overrightarrow{M}(t)$ is a total dipole moment of the system at a given time t, and I(v) is the dipole autocorrelation function, which compares the dipole moment $\overrightarrow{M}(0)$ at time t=0 with the dipole moment $\overrightarrow{M}(t)$ at later times t. n(v) is index of refraction (taken as constant over the frequency range of the experiment), c is the speed of light, k_B is the Boltzmann's constant, and h is the Planck's constant. The absorption coefficients $\alpha(v)$

were calculated with the dipole correlation time averaged over 2,000, 25-ps segments of each MD trajectory.

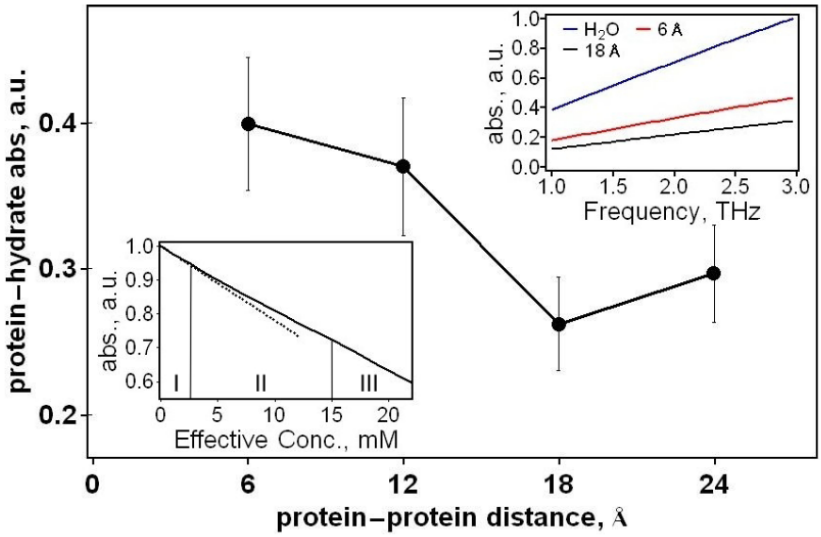


Figure 3.8 : Calculated Terahertz absorbance of λ^*_{6-85} and the first hydration shell

The plot against distance between the protein surfaces shows a non-monotonic trend at 300K. (**Upper Right Inset**) Frequency dependence of the protein-hydration layer absorbance at low (6 Å) and high (18 Å) protein–protein separation, together with the absorbance computed for the same volume of bulk water. (**Lower Left Inset**) Total computed Terahertz absorption against effective concentration of protein. The quasi-linear region at large protein concentration (area III, c>15 mM) reproduces the known behavior, and the nonlinearity at small protein concentration matches the experimental trend measured here (the dashed line is the linear fit to the low concentration trend) [19]. These calculations to interpret our experiments were performed by Matthias Heyden, Xin Yu, and David Leitner.

[Figure 3.8] shows the computed absorbance of the protein and first hydration layer at 2.5 Terahertz as a function of the distance between protein surfaces. Absorption coefficients reported for protein and a hydration shell correspond to the protein and the nearest 3 Å of water molecules. In accord with experiment, we find that the distance between the proteins significantly influences the absorbance of the protein and its first hydration shell (and shells beyond, see [Chapter 3.6.2]). First, the absorbance decreases as the proteins are brought closer together from 24 Å to 18 Å by ~15%. Then the absorbance increases by ~40% when the distance between the protein surfaces shrinks to 12 Å. Finally the absorbance turns over and flattens for the shortest distances, changing little with inter-protein distance, mimicking the concentration-dependent turnover observed experimentally. (There is still a modest increase in the absorbance when the protein–protein separation is reduced further still to 6 Å.) The variation in the absorbance

beyond a protein–protein separation of 18 Å also supports that the hydration shell around each protein extends to at least 9 Å.

This trend is especially strongly pronounced in the calculation because the bulk water, which contributes most at the measured protein concentration, is not included. The trend is less evident if we include the bulk water in the predicted total absorption because the simulation predicts water to have a higher absorbance than low concentrations of protein in water.

Therefore, molecular dynamics simulations of the dipole correlation function of the hydration water supported the hypothesis that the Terahertz absorption of the hydration shell could depend on the distance between the proteins, in agreement with studies by Pettitt and co-workers that show retarded dynamics for water between nearby solutes.[75]

3.4.6 The total Terahertz absorption decreases linearly at moderate and higher concentration

We can computationally estimate the dependence of the total Terahertz absorption coefficient on protein concentration by using the surface-to-surface distances in [Figure 3.8]. A given concentration c corresponds to a distribution of surface-to-surface distances. By considering the concentration dependence of the absorbance of protein and the hydration layer with the bulk water, Monte Carlo sampling of hard-sphere proteins (12.1 Å radius of gyration) yields an estimate for the total absorption as a function of effective concentration [Figure 3.8, Lower Left Inset].

It shows the result, which qualitatively matches the trend in the experimental data at moderate and high protein concentrations: absorbance drops off approximately linearly with increasing concentration, as observed in earlier measurements over a wide range of concentrations [62]. Only by precise measurements of changes at low concentrations does the nonlinear variation, which is a direct probe of the dynamical hydration shell, become apparent. As discussed below, the appearance of a change in the slope of the absorbance vs. concentration at low concentration implies a broad hydration water shell around each

protein, despite the a priori assumption of a single hydration layer made in the preceding computational analysis.

3.4.7 Other evidences of an extended dynamical hydration shell

The unexpected nonlinear absorbance vs. concentration is a collective dynamical property of the protein - hydration water system. Protein–protein distance-dependent changes in the collective dipole moment are evident upon examining the dipole autocorrelation function [Chapter 3.4.5]. While the cross over to the plateau is expected at the concentration of much more than 2.5 mM in the simulation, it actually occurs at less than 1.0 mM for the Terahertz measurements. In the simulation, the hard sphere model was assumed, which does not account for an attractive potential between the proteins. Although lambda repressor shows no signs of irreversible aggregation at concentrations below 20 mM, a nonzero attractive interaction potential between proteins (transient aggregation calculated by simulation based fitting to SAXS experiments, as described in [Chapter 2]) can shift the peak in absorbance toward smaller concentrations, because the actual distance is then smaller than expected for the assumed random distribution, due to the attractive force fields. This explains why the estimated hydration shells size (> 10 Å) from the simulation could be smaller than the hydration shell size observed by the experiments (> 20 Å). However, any long-range interaction cannot explain the observed maximum in the Terahertz absorption, because it would only cause a ‘‘rescaling’’ of the concentration axis. The nonlinearity has to be attributed to the onset of overlapping dynamical hydration layers, which show an increased Terahertz absorption compared with the buffer.

Whereas the existence of hydration shells of over 10Å has not been reported experimentally so far, such large shells containing water dynamically distinct from water in the bulk have been found in earlier molecular dynamics simulations [63, 76]. The heterogeneous rigidity of the water network and its coupling to the protein surface influence the vibrational density of the low frequency modes.[63, 77, 78] Several other

recent studies have also addressed the molecular-level dynamics of hydration layers from the protein surface to the bulk.[79] Using X-ray and neutron diffraction, Head-Gordon and co-workers found for low concentration of the NALMA peptide an additional elastic component is activated, which is attributed to a coupling between inner and outer hydration layers.[80] Molecular dynamics simulations for villin headpiece in aqueous solution yielded a change in the density of water near the protein upon unfolding and a correlation of the water dynamics with the folding process.[81]

In addition to the dipole autocorrelation function and recent works described in the literatures above, a hydration shell corresponding to water dynamics distinct from bulk water can be quantified by the hydrogen bond correlation function, C(t), which yields the probability that a hydrogen bond that exists between two water molecules at a given time, t=0, is present at a later time, t, regardless whether the bond has been broken between 0 and t. The MD simulation of solvated globular λ^*_{6-85} at 27 °C, performed by the Leitner group [19] reveals that the hydrogen bond correlation function for water molecules in 2 Å thick layers of water up to 10 Å from globular λ^*_{6-85} is distinct from the hydrogen bond correlation function computed for bulk water [63], as shown in [Figure 3.9].

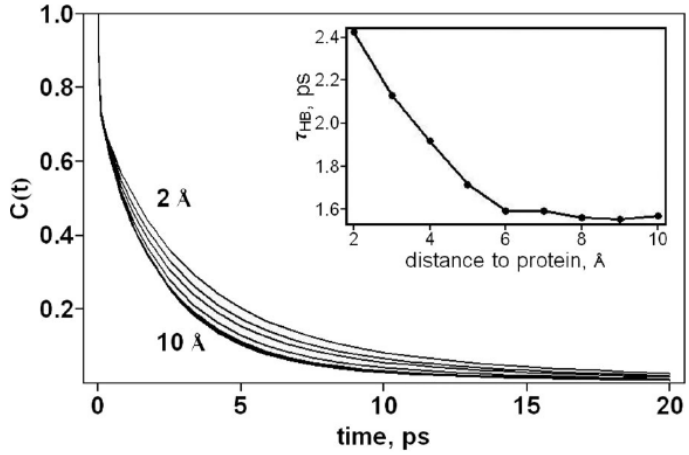


Figure 3.9 : Hydrogen bond correlation function for the water molecules around λ^*_{6-85}

The shown, from top to bottom, within 2 Å of the protein, between 2 and 4 Å, etc., up to between 8 and 10 Å, which appears very close to the bulk water value. (**Inset**) The hydrogen bond lifetimes for water as a function of distance (Å) from the surface of the protein, which is defined as the time at which C(t) is 1/e. [63] These calculations to interpret our experiments were performed by Matthias Heyden, Xin Yu, and David Leitner.

In summary, both experiment and simulations indicate a long-range dynamical hydration shell and reveal the dynamics of the hydration water to be sensitive to the distance between proteins.

3.5 Sequence- and pH-dependent hydration of the lambda repressor[5]

3.5.1 pH-dependent hydration of the lambda repressor

As a global perturbation of protein hydration, first we lowered the pH value from 7.3 to 5, or even down to 2. The absorption of the buffer alone is constant over this pH range. The protein has gone partway through the unfolding transition at pH 2 (as monitored by circular dichroism and fluorescence wavelength shift, shown in supporting materials in ref. [77]). ANS binding to the proteins is enhanced at lower pH (2 and 5), indicating a more exposed hydrophobic surface area.[64]

We observe a strong pH dependence of the Terahertz absorption [Figure 3.10]. At pH 7.3, addition of protein to the buffer increases the absorption coefficient 0.5-1.0 mM concentration, whereas at pH 2 and 5, the protein solution has almost the same or slightly lower absorption coefficient than aqueous buffer. The non-monotonic behavior observed at pH 7.3 cannot be explained by a two-component excluded volume model. In the case of a completely transparent protein which displaces water, we expect a decrease according to the dotted line in [Figure 3.10], but the pH 7.3 data indicate that the absorption coefficient of hydration water is enhanced by the presence of protein. This enhancement at low concentrations indicates a dynamical hydration shell of >10 Å thickness around the protein, as discussed earlier in [Chapter 3.4]. At pH 2, the absorption lies slightly below the dotted line that posits a completely transparent protein,

[5] All protein samples are carefully prepared and provided by the author, and the Terahertz data were collected by Simon Ebbinghaus in the Havenith group, at Ruhr-University-Bochum, Germany.

indicating hydration water with an unusually low absorption coefficient. The pH 5 data follow the dotted line more accurately.

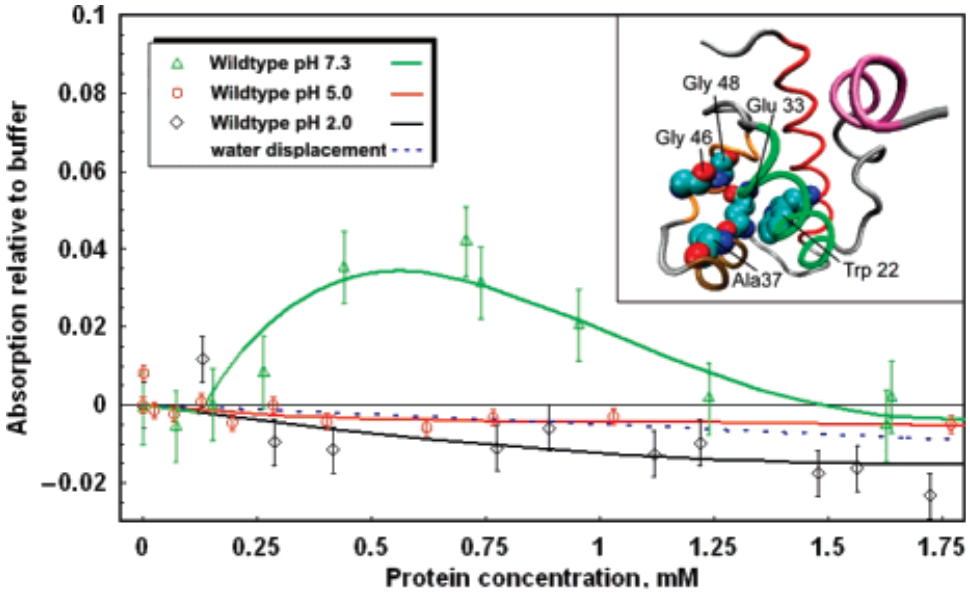


Figure 3.10 : Terahertz absorption of λ^*_{6-85} at pH 2.0 / 5.0 / 7.3

Difference in the integrated Terahertz absorption coefficient (2.1-2.8 Terahertz) of λ^*_{6-85} at pH 2.0, pH 5.0, and pH 7.3 relative to $\alpha_{rel} = [\alpha_{protein} - \alpha_{bulk}]/\alpha_{bulk}$ plotted against concentration. Shown is the average of several subsequent measurements at the same concentration along with the statistical error. The main error source is the refilling of the sample cell. Further details of the experimental setup can be found in ref. [74]. The temperature is kept at 20°C. The inset shows the structure and the mutation sites. The absorbance for the native protein (pH 7.3) depends nonlinearly on concentration in this region, indicating overlapping hydration shells. In contrast the concentration dependence of the Terahertz absorption of the destabilized protein (at pH 2 and 5) resembles the predicted decrease due to the replacement of water molecules by the proteins, described in the two-component excluded volume model.

To complement the experimental data, our collaborators, M. Heyden, X. Yu, D. Leitner, and M. Havenith have studied the approximate dynamics of the protein and explicit solvent water by molecular dynamics (MD) simulation. They calculated the predicted average lifetimes of hydrogen bonds for λ^*_{6-85}, and it shows that water molecules around the denatured state show retardation of the dynamics, caused by the exposure of hydrophobic residues of the denatured protein to the water. [77] The more exposed hydrophobic residues significantly change the hydration dynamics and induce negative THz absorption in surrounding water molecules.

3.5.2 Sequence dependent hydration of the lambda repressor

To study a site-specific hydration effect on the Terahertz spectrum, I substituted Gln33 for Tyr by site-directed mutagenesis, replacing the highly polar glutamine side chain ($CH_2CH_2CONH_2$) by a less polar aromatic side chain. When coupled with Ala-Gly mutations (A37G/A49G) that greatly destabilize the protein,[82] the Tyr mutant shows a concentration dependence similar to the low pH proteins, with only a remnant of a concentration maximum. When coupled with a helix-stabilizing mutation (G46A/G48A), about half the maximum in absorption relative to buffer is restored when compared to pH 7.3. Thus a quadruple mutation (A37G/A49G to G46A/G48A) that stabilizes helices in λ^*_{6-85} is not sufficient to completely offset the effect induced by a single point mutation at position 33.

The results are summarized in [Figure 3.11]. Terahertz absorption can thus be used in conjunction with site-directed mutagenesis to probe local interaction of protein surfaces with their solvent shells.

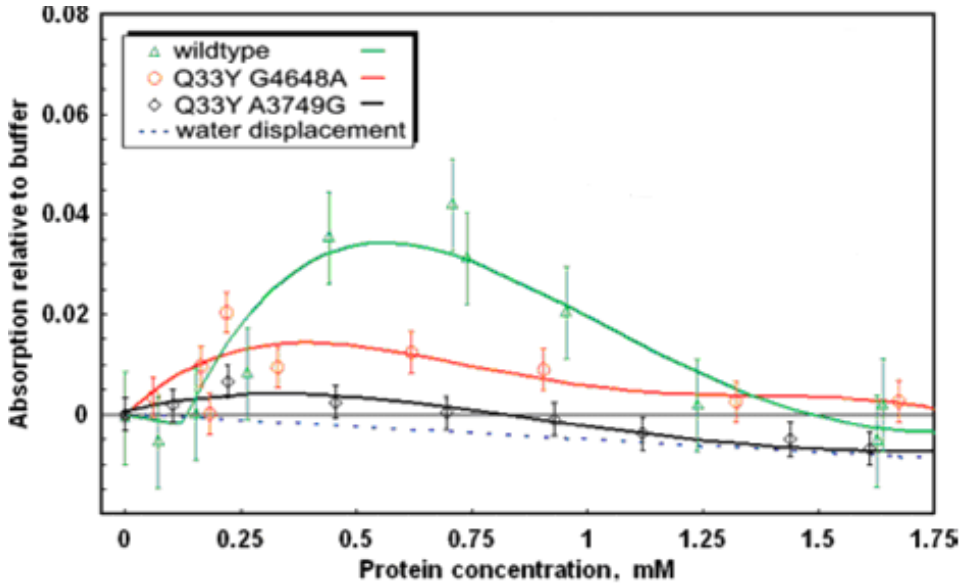


Figure 3.11 : Terahertz absorbance of λ^*_{6-85} and its mutants at pH 7.3

A comparison of the integrated Terahertz absorbance (between 2.1 and 2.8 Terahertz) of the pseudo-wild-type lambda repressor with two mutants of the protein at pH 7.3. The nonlinear concentration response is most pronounced for the wild type. It is less significant for the helix-stabilized mutants. The mutant Q33Y/A37G/A49G deviates the least from a simple solvent displacement model (dotted line).

In summary, we have shown that global perturbations of the protein hydration shell by pH and local perturbation by surface site-specific mutation both produce significant changes in the terahertz absorption spectrum of aqueous protein. Such changes can be used in the future as sensitive probes of protein-solvent dynamics, opening up the possibility of using Terahertz absorption as a probe for protein folding kinetics and functional dynamics measurements. The development of quantitative models for the Terahertz spectra will make it possible to understand local hydration of proteins at the molecular level.

3.6 The effect of protein flexibility on the dynamical hydration shell of ubiquitin

3.6.1 Results

I summarize the results of the measured changes in the Terahertz absorption of all five protein variants in [Figure 3.12][6]. For reference, the dotted line also shows what would be expected for a simple two-component model with protein and bulk water only. Similar to λ^*_{6-85}, the measured Terahertz absorption of all five proteins deviates strongly from a linear decrease which is predicted in [Equation (3.4)]. The wild-type and pseudo-wild-type (containing a tryptophan) in particular deviate strongly from a linear concentration dependence. The mutants whose fluorescence indicates higher flexibility deviate less from bulk buffer absorption, but still significantly outside the measurement uncertainty shown by the error bars. Note that the ranking from highest to lowest flexibility based on our fluorescence measurements is Ub* I61A > Ub* V26A I61V > Ub* V26A ≈ Ub* > Ub, [66] whereas the ranking from lowest to highest deviation in the Terahertz absorption is Ub* V26A I61V ≤ Ub* V26A < Ub* I61A < Ub* < Ub. (See [Table 3.1] and [Chapter 3.3.2] for the abbreviation of the ubiquitin mutants.)

[6] All protein samples are carefully prepared and provided by the author, and the Terahertz data were collected by Benjamin Born in the Havenith group, at Ruhr-University-Bochum, Germany

When extrapolating the Terahertz absorption coefficient towards very high protein concentrations we find slight differences between the different mutants. Whereas the wild-type shows the highest net absorption in this spectral range, Ub* and Ub* V26A are found to have a similar but slightly smaller Terahertz absorption coefficient at the highest measured protein concentration. Both exceed that of Ub* V26A I61V and Ub* I61A, which approach the value expected for a completely transparent sphere of the volume of the protein. The concentration at which the maximum in the Terahertz absorption for all ubiquitin and mutants is found lies around 1.25-1.5 mM, which is higher than the maximum in the Terahertz absorption for the five helix bundle λ^*_{6-85} at ca. 0.6 -0.7 mM concentration. This indicates that ubiquitin has a smaller dynamical hydration shell than λ^*_{6-85}.

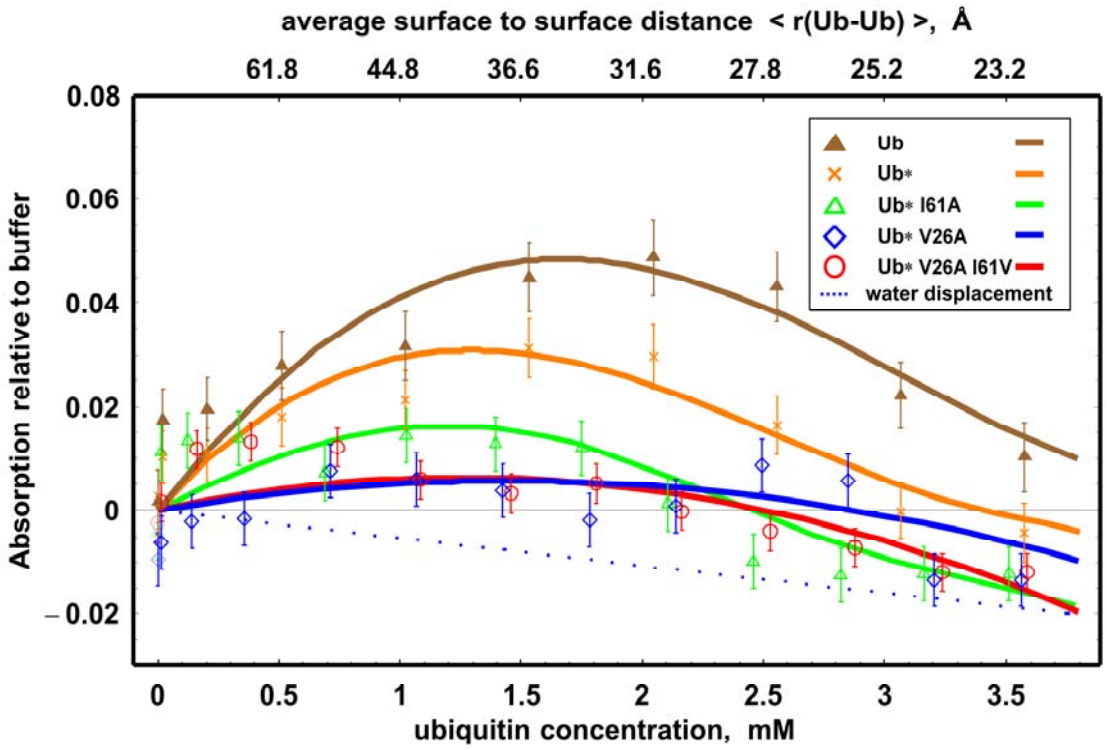


Figure 3.12 : Terahertz absorption of ubiquitin and its mutants

Integrated Terahertz absorption coefficient (between 2.1-2.8 Terahertz) of the protein as a function of protein concentration. Displayed is the result for ubiquitin wild-type, the ubiquitin pseudo-wild-type and three ubiquitin mutants relative to bulk water. The measurements ware carried out at (20±0.5) °C and at pH 4.8. The dotted line shows the predicted decrease in case that the protein does not contribute to the total Terahertz absorption, but is just displacing water molecules (two component excluded volume model).

3.6.2 A fit to the three component excluded volume model

The concentration at which the onset of non-linearity occurs is directly correlated with the smallest concentration at which the dynamical hydration shells start to overlap. [Figure 3.13] shows a fit of the Ub data to [Equation (3.5)], which is performed by Benjamin Born in Ruhr-University-Bochum, Germany.

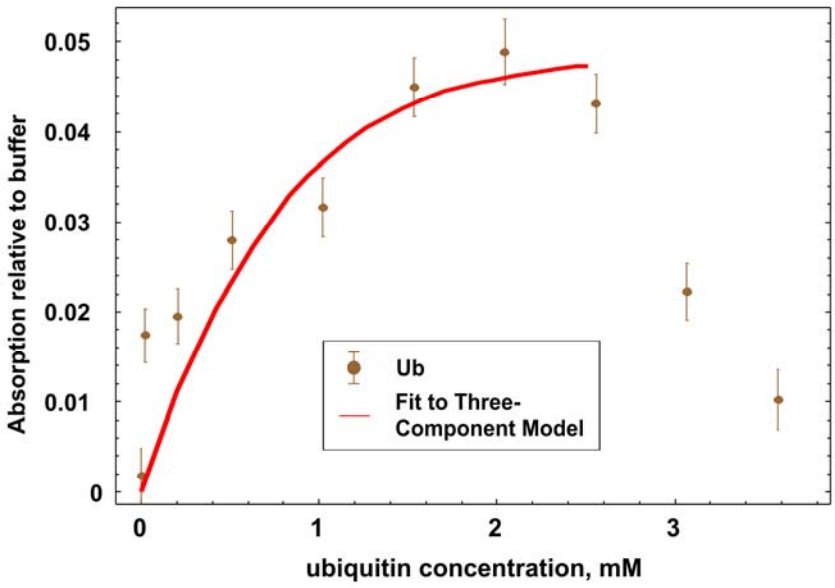


Figure 3.13 : A fit to the three-component model for wildtype ubiquitin

Wild-type ubiquitin Terahertz absorption fitted to a three component Monte Carlo model that takes into account overlapping hydration water shells at higher concentration.[83] This fit to interpret the experiment results was performed by Benjamin Born, Ruhr-University-Bochum, Germany.

He has restricted the fit to concentrations below 2.5 mM because the three-component model cannot account for the full decrease of absorption observed at higher concentrations. The truncated bulk solvent and hydration sphere volumes required by [Equation (3.5)] were simulated by a Monte Carlo distribution of globular proteins with spherical hydration shells whose diameter can be adjusted.[83] We can then directly deduce the average size of the hydration shell from the concentration at the point of turnaround in the Terahertz absorption.

He has added these calculated protein surface – protein surface distances as a further variable to the x-axis of [Figure 3.12]. For ubiquitin the molar concentration of

the maximum is around 1.5 mM, which corresponds to an average protein - protein center of mass distance D of ≈ 66.6 Å.

$$\frac{4\pi}{3}D^3 \simeq \frac{10^{-3}m^3}{0.0015\times(6.02\times10^{23})} \tag{3.10}$$

If we take a surface radius of 15 Å for the bare ubiquitin, the average distance between the protein surfaces at 1.5 mM is 36.6 Å.

$$\begin{aligned} D &= 66.6\ \text{A} \\ &= 2\text{R}_{\text{shell}}\left(\sim 18.3\ \text{A}\right) + 2\text{R}_{\text{surface}}\left(\sim 15\ \text{A}\right) \end{aligned} \tag{3.11}$$

We can then directly deduce the average size of the hydration shell, which corresponds to R_{shell} at the concentration at the point of turnaround in the Terahertz absorption to be 18.3 Å. This number still exceeds by far the estimated size of the sterically bound first hydration shell (~3 Å).

However, it must be noted that the three-component model oversimplifies the situation, and does not provide a quantitative fit over the full concentration range. As seen in [Figure 3.13], the fitted function does not drop off rapidly enough at higher concentrations. Proteins are rather large molecules compared to disaccharides,[74] and at high enough concentration, a large fraction of the hydration water lies in the hydration shells around two or more proteins. Such multiple hydrated waters may differ from hydration water around a single protein. The overestimate of absorbance in [Figure 3.13] at higher concentrations would in fact indicate that water interacting with multiple proteins absorbs less than water interacting with one protein. For λ-repressor in [Chapter 3.4] we also showed that the assumption of a single hydration shell, but constant absorption coefficient is too simple for proteins. The molecular dynamics simulations performed by our collaborators [Chapter 3.4.5] supports a more complex Terahertz absorption which depends on the protein-protein distance.[19]

3.6.3 Terahertz vs. Fluorescence spectroscopy: *the tryptophan effect*

Fluorescence and Terahertz spectroscopy report differently on the flexibility of the protein. Tryptophan is a local probe, and one would expect it to be most sensitive to the environment near the side chain. Terahertz spectroscopy of the dynamical hydration shell is a global probe that averages over the entire protein surface. This is borne out by the ranking reported in the results. In [Figure 3.3] on the right, the fluorescence measurements show large deviations from the pseudo-wildtype when residue isoleucine 61, adjacent to the tryptophan, is mutated to alanine or valine. On the other hand, mutation of the remote residue valine 26 to alanine has almost no effect on the fluorescence, even though this mutation is more destabilizing to the protein overall. In contrast, Terahertz spectroscopy shows the largest deviation from Ub^* for the more disruptive core mutation Val26Ala and for the corresponding double mutant, but a smaller effect for the less disruptive Ile61Ala mutation of a near-surface residue. Thus tryptophan detection emphasizes the side chain truncation near the tryptophan residue, while Terahertz spectroscopy emphasizes the more destabilizing truncation of a core residue.

The tryptophan probe itself can be evaluated further by our Terahertz measurements. The interesting question is: how much does insertion of a tryptophan side chain modify the hydration dynamics? As we can see from [Figure 3.12], the introduction of the fluorescent tryptophan in the Phe45Trp variant Ub^* shows a peak at a slightly lower concentration (1.25 mM) than the wild-type Ub (1.5 mM), and we obtain a statistically significant reduction of the Terahertz absorption at the concentration of the maximum. Thus the tryptophan probe has an impact on the absorbance of the hydration water network Terahertz vibrations, and thereby on the fast hydration dynamics. This is an important consideration for fluorescence studies that depend on the insertion of tryptophan probes at various sites, and which generally assume unperturbed hydration dynamics upon insertion. However, the good news for fluorescence studies of hydration water is that phenylalanine to tryptophan replacements cause smaller changes in the absorbance characteristics of the hydration water than any of the other mutations, e.g. the

side chain truncations.

3.6.4 Hydrophobicity significantly affects hydration water structure

In [Chapter 3.5.1], a complete absence of the maximum was found for denaturated λ-repressor at pH 2, which is more flexible than the native structure.[77] ANS binding indicated that this reduction might be associated with the increased exposure of more hydrophobic sites which affect the water in its hydration shell. [64, 77] The more the protein loses its structures and gains flexibility, the more hydrophobic residues are exposed to water molecules, as shown in [Figure 3.14]. Thus increased surface hydrophobicity of the mutants is a candidate for changes in hydration water structure that leads to the smaller bulk water-like Terahertz absorption, as compared to the Ub and Ub* proteins. Based upon MD simulations [84], we propose that the solvent exposed hydrophobic side chains induce a negative Terahertz absorption coefficient in their surrounding, whereas hydrophilic parts lead to an increase in the hydration water compared to bulk water.

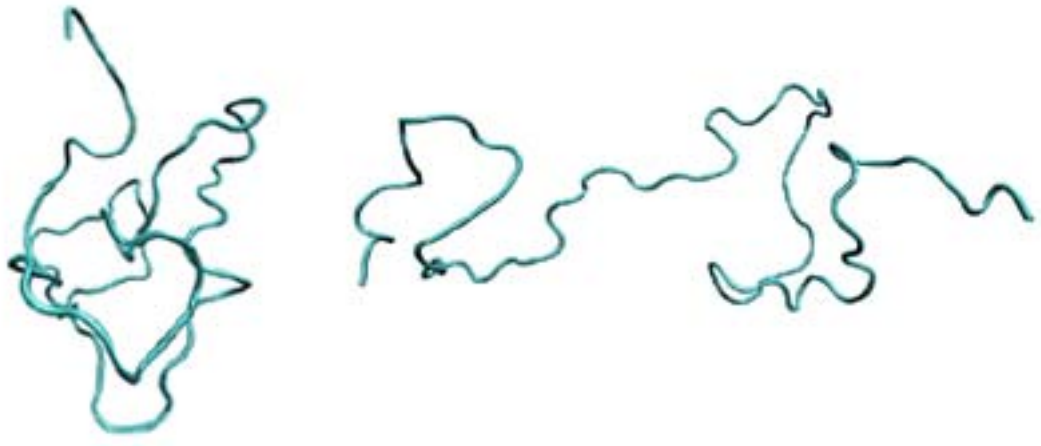


Figure 3.14 : VMD visualization of a partially folded (left) and a fully unfolded (right) Ubiquitin

The more the protein loses its structures and gains flexibility, the more hydrophobic residues are exposed. 1UBQ structure from PDB Databank [85] using VMD visualization to show its 3D shapes [37]

To test this idea further, we measured the absorption coefficient of denatured ubiquitin as a function of concentration. [Figure 3.15] shows that ubiquitin, like

λ–repressor, has a signature very close to the two-component bulk water model of [Equation (3.4)] once it has been denatured. This supports the idea that in case of partial unfolding the increased exposure of the hydrophobic core leads to a decrease of the initial maximum in Terahertz absorption at 1.5 mM. The resulting curve resembles now that of bulk water with the protein displacing a water volume of 14400 $Å^3$ multiplying number of proteins.

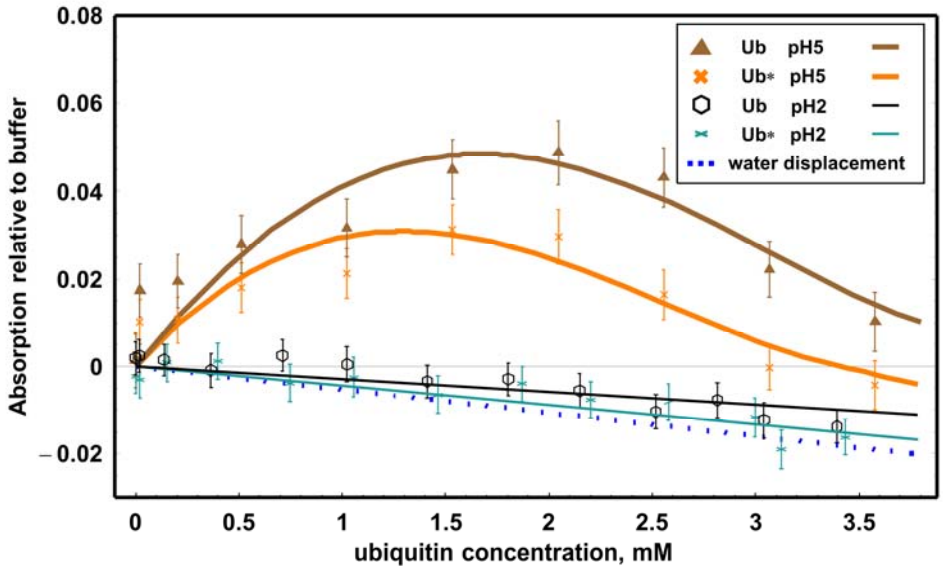


Figure 3.15 : Terahertz absorption coefficient of Ub and Ub* at pH 2 and pH 4.8

Integrated Terahertz absorption coefficient (between 2.1-2.8 Terahertz) of Ub and Ub* as a function of protein concentration at pH 2 and pH 4.8. The measurements were carried out at (20±0.5) °C and at pH 4.8. The dotted line shows the predicted decrease in case that the protein does not contribute to the total Terahertz absorption, but is just displacing water molecules (two component excluded volume model).

3.7 Summary

In summary, we find that Terahertz absorption spectroscopy provides a sensitive tool to probe the fast hydration water dynamics around proteins. At low concentrations we find a non-linear absorption dependence on concentration. This nonlinearity indicates a long range (up to ~20 Å from the protein surface) influence on the hydration dynamics, corresponding to 6 hydration water layers (ca. 3 Å average extension per water molecule).

This long range influence is sensitive to changes in the overall flexibility. The Terahertz absorption at low concentrations is significantly altered when the protein is partially unfolded.

3.8 Acknowledgement

I especially appreciate Simon Ebbinghaus and Benjamin Born for the cordial welcome and the sharing of their instruments and experimental results when I visit Ruhr-University-Bochum, Germany. I was very lucky to have chances of collaborating with these nice people. I wish to thank E. Bründermann and G. Schwaab for the initial instrument development and many helpful discussions, and software development. I gratefully acknowledge financial support from the Human Frontier Science Program.

Chapter 4 Real-time detection of protein-water dynamics upon protein folding by KITA (Kinetic Terahertz absorption) spectroscopy

4.1 Abstract

Kinetic Terahertz absorption (KITA) spectroscopy is introduced to study folding of solvated biomolecules. KITA is particularly sensitive to protein-hydration water dynamics. We apply KITA to the refolding kinetics of ubiquitin and of three side chain truncation mutants designed to disrupt the hydrophobic core and increase overall protein flexibility. KITA results are compared to small angle X-ray scattering, tryptophan fluorescence, and circular dichroism. The KITA signal rapidly relaxes to the native protein's value, on the same millisecond time scale on which secondary structure formation is detected by circular dichroism. Both processes are much faster than acquisition of native-like fluorescence. We propose that KITA monitors the rearrangement of hydrogen bonding during secondary structure formation, and suggest future experimental tests and applications to folding dynamics with this new technique.

4.2 Introduction

Recently, there has been a growing interest in probing not just the dynamics of self-assembling macromolecules, but the dynamics of their hydration shells as well. Dielectric, Raman and fluorescence spectroscopies, NMR, neutron scattering and crystallography all provided insights, but only Terahertz absorption spectroscopy (wavelength range 0.1-1 mm; 1 Terahertz = 1 ps^{-1}) probes the picosecond solvent dynamics directly over any desired time scale, and is sensitive to hydration layers far from the molecular surface.[63]

Protein folding is a self-assembly process in which solvent motions play a critical role. The free energy contributions of the protein and of the hydration water are comparable during folding,[86] and water dynamics are perturbed by the protein beyond two hydration layers.[83, 87] Yet folding has been probed in the past mainly with an emphasis on the backbone and side chains of the protein itself. Can we directly probe solvent reorganization during secondary structure or hydrophobic core formation?

Terahertz sources have become powerful enough to study directly the absorption spectroscopy of biomolecules in aqueous buffer.[62, 74, 88-91] We recently showed that Terahertz absorption is sensitive specifically to hydration water around proteins.[19, 77] At the same time, time-domain Terahertz spectroscopy has been applied in absorption and emission to study picosecond dynamics on the time scale of the Terahertz pulse itself,[92, 93] and Terahertz absorption has been used to monitor slow kinetics.[94] Terahertz absorbance probes dynamics on the 10^{-12} second (picosecond) time scale, ideal for monitoring translational/rotational/vibrational dynamics of the water network near the protein surface, notably hydrogen bond rearrangements.

In our previous work, for the five-helix bundle λ^*_{6-85} and for ubiquitin, we observed excess absorption of 2.5 Terahertz light by millimolar protein solutions, compared to the buffer or the protein alone. We showed that altered water dynamics within hydration shells of up to 15 Å in thickness account for the excess absorption.[19, 77, 83] We suggested that the excess absorption of hydration water at 2-3 Terahertz occurs because the protein-water coupling induces a shift of absorbance from sub-Terahertz to higher frequency modes.[19] At even higher concentrations of proteins, Plaxco and coworkers determined that Terahertz absorption decreases quasi-linearly,[62] and our measurements agree with this result.

Here we find that millimolar protein solutions indeed absorb less than buffer in the 0.2 – 0.8 Terahertz region, which is in agreement with this suggestion. We then use the change in Terahertz absorbance to monitor folding kinetics. Here we introduce kinetic Terahertz absorption (KITA) for hydration dynamics during folding. KITA provides a direct window on protein-solvent rearrangements during folding, such as the breaking of backbone-water hydrogen bonds and their replacement by backbone-backbone hydrogen

bonds.[95] KITA monitors the changing Terahertz electric field pulse shape on the picosecond time scale Δt, as a chemical reaction proceeds on a longer time scale t, up to many seconds. We apply KITA to measure the changing protein-hydration water dynamics during the fast refolding of ubiquitin. We have chosen human ubiquitin with a Phe45Trp mutation (Ub*) for our first KITA folding study. Ub* is a 76 residue predominantly β-sheet protein, which has long been used as a prototype for folding kinetics studies.[96] We previously probed ubiquitin folding by circular dichroism (sensitive to secondary structure formation), fluorescence (sensitive to dehydration around an engineered tryptophan) and small angle X-ray scattering (sensitive to the radius of gyration).[71, 97] These studies provide an opportunity to compare KITA with a number of existing spectroscopic probes of folding.

Highly probe-dependent refolding kinetics are observed. The folding kinetics detected by KITA are compared to small angle X-ray scattering (SAXS), tryptophan fluorescence, and circular dichroism (CD), revealing that in the 0.1-1 Terahertz range, the hydration dynamics are coupled to secondary structure formation (including a switch from solvent-protein towards more protein-protein hydrogen bonds) and to protein compactification, whereas formation of native-like tertiary structure around the tryptophan takes place on a thousand-fold slower time scale. We find that the change in Terahertz absorption, which monitors collective rearrangements of the protein chain and hydration water, has a millisecond response. On a similar time scale, we observe significant changes in secondary structure content and protein compactness. In sharp contrast, tryptophan fluorescence takes at least a second to switch from the denatured to the native state. Thus rapid adaptation of the hydration water around a protein occurs long before hydrophobic residues are packed into a native-like environment. Our finding supports the hypothesis of Frauenfelder and coworkers that early protein folding protein dynamics is slaved to hydration dynamics.[47]

To extract further structural information, we also monitored the absorption of three mutants of Ub*, involving side chain truncation of fully or mostly buried aliphatic residues (Valine, Isoleucine) so as to minimize any change in the interactions of the native protein with hydration water. The early folding dynamics monitored by KITA are not greatly affected by mutations that affect the core packing of the native state.

Kinetic Terahertz absorption promises to be a useful tool for studying the dynamics of the hydration environment around proteins. Recent simulations of absorption spectra of hydrated proteins, achieved by monitoring the picosecond rearrangement (1/Terahertz = 10^{-12} seconds) of dipole moments in molecular dynamics simulations, [19, 63] also show the sensitivity of the water hydration network to Terahertz absorption. It will be very interesting to compare such simulations for unfolded, partially folded and native states of proteins in the future, to go hand-in-hand with KITA experiments of protein folding.

4.3 Materials and methods

4.3.1 Protein sample

Ubiquitin is a small predominantly β-sheet protein with 76 residues (MW 8.5 kDa; see [Figure 4.1]). The plasmids for ubiquitin mutants were made as described in ref. [66] from the original Ub* plasmid (provided by Tracy Handel), [72] which has a Phe45Trp mutation to introduce a fluorescent marker. We studied two single point mutants (I61A, V26A) and one double mutant (V26A I61V) to examine the effect of flexibility caused by side chain truncation. (Refer to [Table 3.1] and [Chapter 3.3.2] for the abbreviation of the ubiquitin mutants.) Protein flexibility was previously shown to reduce the Terahertz absorption of native ubiquitin in the 2.5 Terahertz region.[83]

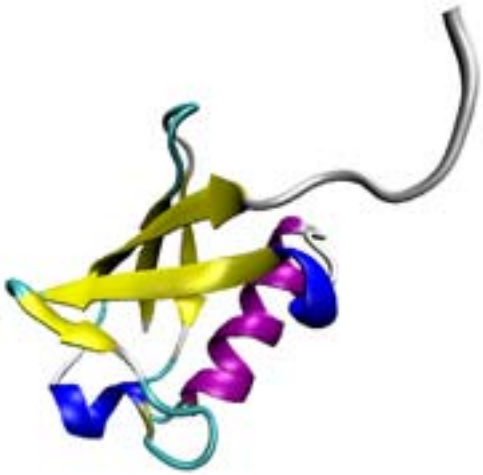


Figure 4.1 : VMD visualization of Ubiquitin and its structures by color

1UBQ structure from PDB Databank,[85] using VMD visualization to show its structures by color[37]

Plasmids for each mutant were inserted into the pET-15b vector and expressed in Rosetta TM (DE3) pLysS cells (Novagen Inc). After cell growth in LB broth at 37 °C for 8 hours, we performed an induction with IPTG and kept cells at 25 °C for 12 hours. Cells were lysed with a French press. Collected supernatants were bound to a CM-52 cation exchange column and eluted with a linear salt gradient from 0 to 1 M NaCl for purification. Flow-through containing ubiquitin was collected and additional purification was performed with Amicon 3 kDa and 30 kDa membranes (Fisher Scientific). The purity of ubiquitin mutants was checked by electrospray ionization mass spectroscopy and SDS-PAGE. Final protein concentrations were determined by UV absorption spectroscopy (Shimadsu UV-1650 PC) at 280 nm.

Protein was dissolved in a 45%/55% by volume ethylene glycol/water buffer with 40 millimolar sodium phosphate at pH 5.9. This allowed cooling of the solutions to as low as –28 °C for comparison with previous X-ray scattering experiments. 6 molar of guanidine hydrochloride was added to denature protein before mixing in the stopped-flow. After 1:6 mixing, and in the reference buffers used for subtraction, the guanidine hydrochloride concentration was 0.86 M. The denaturation curves by guanidine hydrochloride are shown in ref. [66].

4.3.2 KITA measurement details

Our apparatus is illustrated in [Figure 4.2] and [Figure 4.3]. Terahertz pulses pass through a stopped-flow cell, where a mixer combines denatured ubiquitin with denaturant-free buffer to start refolding, The shape of the transmitted Terahertz electric field is detected using a ZnTe crystal and a 800 nm gating pulse delayed by Δt. The difference ΔE of the electric field between buffer and denaturant-free 1.5 mM protein solution is shown. For kinetics, the Terahertz pulse is detected near the maximum electric field, and the mixer is scanned in time t with respect to the Terahertz pulse.

It resolves the Terahertz pulse with sub-ps time resolution, and measures with millisecond time resolution the changes in the pulse caused by changing absorption of hydration shells and their associated proteins during refolding initiated by a mixer. Pulses spanning the 0.1-1 Terahertz frequency range were used. By scanning the time delay of the Terahertz pulse relative to the gating pulse, the Terahertz electric field is mapped out precisely. By changing the "kinetic" time between stopped-flow and Terahertz pulse, the kinetics of folding are mapped out.

Stopped flow kinetics were used to initiate refolding of Ub* and its mutants for the Terahertz detection. Stopped flow kinetics was measured by 1:6 mixing from 6 to 0.86 M guanidine hydrochloride in stopped-flow instruments (Unisoku, Ltd.). A buffer containing protein and denaturant (6 M guanidine hydrochloride) is mixed with a denaturant-free buffer, and then injected into the observation cell. 1:6 mixing in two stopped-flow instruments (both Unisoku) resulted in a final guanidine hydrochloride concentration of 0.86 M, and in a final protein concentration of 1.5 mM (Ub*) or 1.0 mM (mutants of Ub*). At this denaturant level, all mutants fold to the native state. The dead time of the instrument ranged between 6 ms and 50 ms, depending on the temperature, instrument configuration and solvent conditions. The KITA stopped flow observation cell has a pathlength of 0.5 mm, with 50 μm z-cut quartz windows.

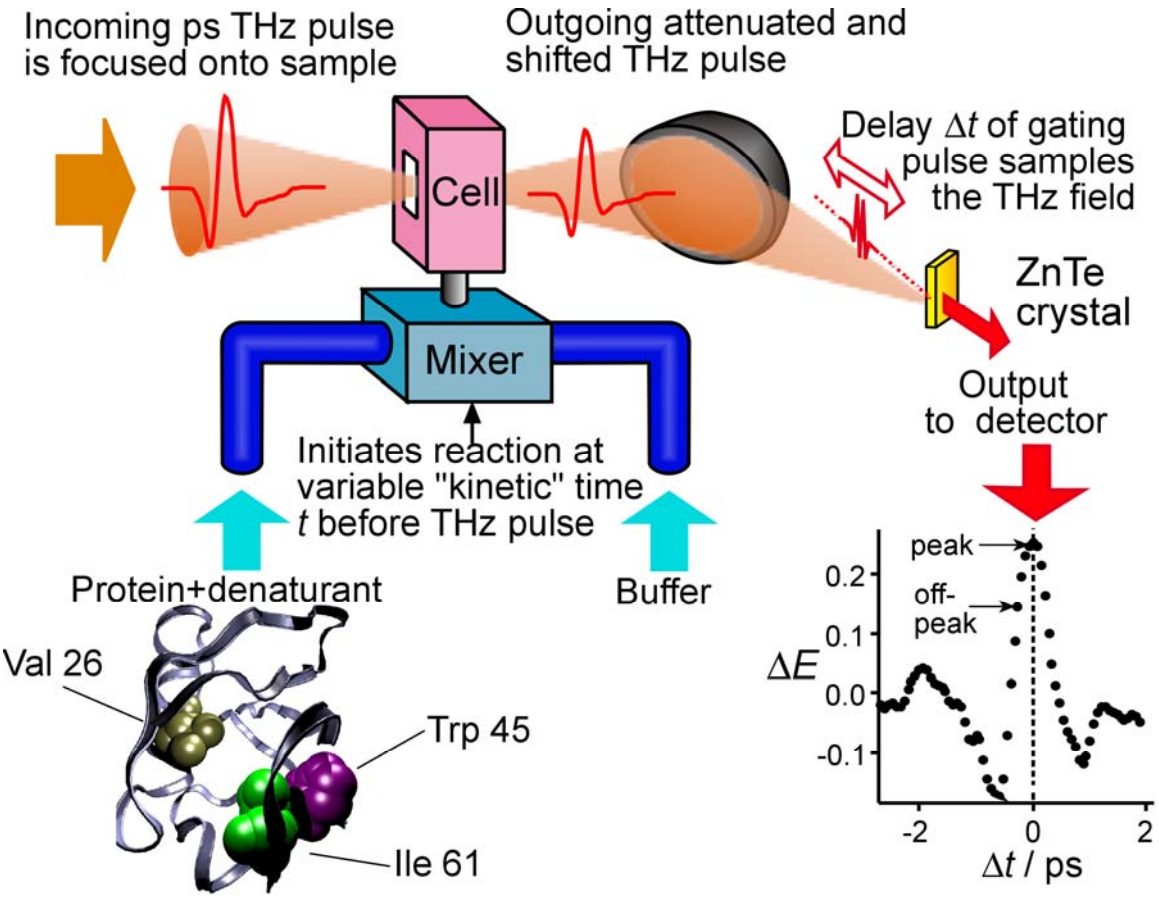


Figure 4.2 : Data collection setup for KITA

Terahertz pulses (orange) are focused into a KITA stopped flow observation cell (pink). A mixer (blue) combines denatured protein solution with buffer to initiate refolding, then injects the protein sample into the cell (1UBQ structure from PDB Databank,[85] using VMD visualization to show mutation sites [37]). The transmitted Terahertz pulse, which is now attenuated and delayed in time, is refocused onto a zinc telluride crystal for detection. There the Terahertz pulse electro-optically modulates a 800 nm laser pulse, imprinting its amplitude onto the laser pulse. By delaying the Terahertz pulse relative to the laser pulse, the electric field of the Terahertz pulse is mapped out. For kinetics, the delay is fixed at or near the maximum Terahertz electric field. To increase sensitivity, the input Terahertz pulse amplitude is modulated at 40 kHz and a lock-in amplifier detects signal only at 40 kHz, providing efficient noise suppression by phase-sensitive detection

Terahertz pulses of about 4 picosecond total duration (600 femtoseconds full width at half maximum) and spanning the 0-1 Terahertz frequency range [Figure 4.5] are generated by photoconductive switching of near-infrared pulses from a Ti:sapphire laser on a Tera-SED low temperature grown gallium arsenide photoconductive emitter made by Gigaoptics™. The specifications of the input pulses are 800 nm wavelength, 20 femtosecond duration at 500 mW average power and 92 MHz pulse repetition rate. The average Terahertz output power is about 10 μW, in picosecond duration pulses at 92 MHz

repetition rate. The Terahertz pulses are focused by an off-axis parabola into the stopped-flow cell. The transmitted Terahertz pulses, attenuated and shifted after protein folding is initiated by the stopped-flow, are then refocused onto a 1 mm thick zinc telluride crystal cut at <110> orientation. To trace out the Terahertz electric field, another 800 nm pulse, derived from the same Ti:sapphire laser is also focused onto the crystal. The interaction of the two pulses generates a gated output signal at 800 nm that is detected by a Nirvana autobalanced photo detector (New Focus). By scanning the time delay of the Terahertz pulse relative to the 800 nm reference pulse on a translation stage (≈0.6 mm per picosecond), the electric field is mapped out precisely, as shown in [Figure 4.5] and [Figure 4.4].[98]

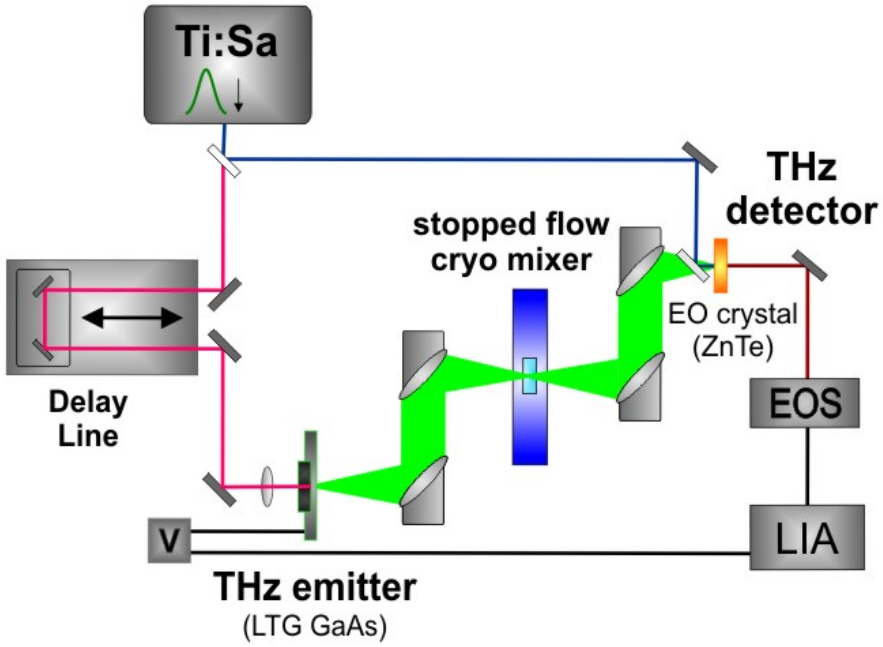


Figure 4.3 : KITA setup overview[7]

To detect this signal with the highest possible sensitivity, the Terahertz pulse was amplitude-modulated at 40 kHz by applying a ±50 V square wave to the photoconductive emitter. The detector current was fed into a lock-in amplifier (Signal Recovery 7265 DSP) with 30 dB input gain set to the same reference frequency. The time constant of the

[7] By courtesy of Benjamin Born, Ruhr-University-Bochum, Germany

lock-in amplifier is set to 5 milliseconds, faster than the dead time of the instrument or the fastest kinetic transients observed. The resulting electric field as a function of delay time, or at a fixed delay time but as a function of "kinetic time" after the stopped-flow, was accumulated into a computer using the National Instruments LabView software.

The relative humidity around the stopped-flow cell was kept below 9% at 19 °C by a stream of nitrogen gas, to avoid attenuation of the Terahertz pulses by water vapor, and to prevent condensation from forming on the optical windows of the stopped flow cell.

4.3.3 Fluorescence kinetics measurements

Stopped flow kinetics were used to initiate refolding of Ub* and its mutants for the fluorescence measurement. Stopped flow kinetics was measured by 1:6 mixing from 6 to 0.86 M guanidine hydrochloride in stopped-flow instruments (Unisoku, Ltd.). We used 280 nm UV laser pulses to excite the tryptophan residue of Ub* and collected integrated fluorescence (λ>320nm) with a photomultiplier. 280 nm cut-off filters (Schott WG320 and Hoya U-360) were used to selectively collect fluorescence and block UV excitation pulses. Time-evolution of fluorescence was recorded by a LeCroy 9384L digitizer coupled to the photomultiplier by an SR 570 current preamplifier (Stanford Research System). Final protein concentration was measured as 29 μM after 1:6 mixing. An observation cell, with a pathlength of 1 mm and 50 μm sapphire windows was used for the fluorescence measurement.

4.3.4 Statistical analysis

The Terahertz and fluorescence data were fitted to single exponential models ($y=A_0+A_1 \exp[-kt]$) and double-exponential models ($y=A_0+A_1 \exp[-k_2t]+A_2 \exp[-k_2t]$) by a Levenberg-Marquart algorithm using equal weights for all data points shown in [Figure 4.7], [Figure 4.9] and [Figure 4.10]. In all cases 2σ fitting uncertainties were below 15%

of the parameter values. In the case of Terahertz kinetic times shown in [Figure 4.7], [Figure 4.9] and [Figure 4.10], the actual error is limited by the dead-time of the stopped flow instrument at low temperature, which prevents data collection < 50 ms. The relaxation times listed are thus most likely lower limits on the actual relaxation times.

4.4 Results

4.4.1 Protein and hydration water absorb less than buffer at 0.2-0.8 THz

For the fluorescent Phe45Trp (F45W) mutant of ubiquitin (Ub*), we previously observed excess absorption of 2.5 Terahertz light in 0.5-1.5 millimolar protein solutions, compared to the buffer or the protein alone. We showed that altered water dynamics up to 18 Å from the protein surface accounts for the excess absorption. [19, 77, 83] We suggested that the excess absorption of hydration water at 2-3 Terahertz occurs because the protein-water coupling induces a shift of absorbance from sub-Terahertz to higher frequency modes. [19]

Here we find that ubiquitin solution absorbs less than buffer in the 0.1 – 1 Terahertz region, in agreement with this suggestion. [Figure 4.4] traces out the difference in the Terahertz electric fields between a 1.5 mM solution of Ub*, and the water/ethylene glycol buffer alone. Fourier transforming the Terahertz electric fields from the time to the frequency domain yields the transmitted intensity for protein solution and pure buffer, shown in [Figure 4.5].

Our pulse covers the spectrum from 0.2-0.8 Terahertz, peaking at about 0.5 Terahertz. The protein and its hydration water typically absorb 10-20% less than the bulk water they replace. The net difference between buffer and protein is nearly constant from 0.2-0.8 Terahertz, so one would expect KITA-detected kinetics not to be wavelength-sensitive in this range.

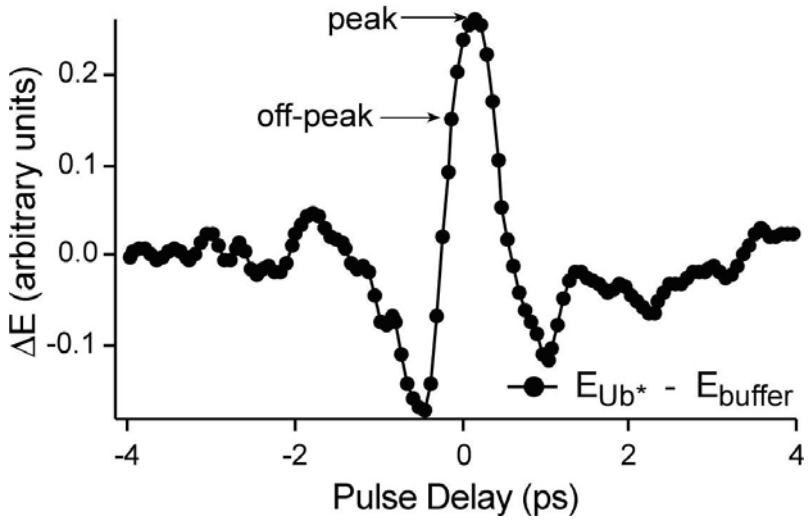


Figure 4.4 : Net Terahertz electric field of Ub* as a function of time

"Peak" and "off-peak" label two times where the Terahertz pulses were sampled for probing kinetics. "Pulse delay" is the variable delay time between the Terahertz pulse and an 800 nm femtosecond laser pulse that maps out the Terahertz pulse.

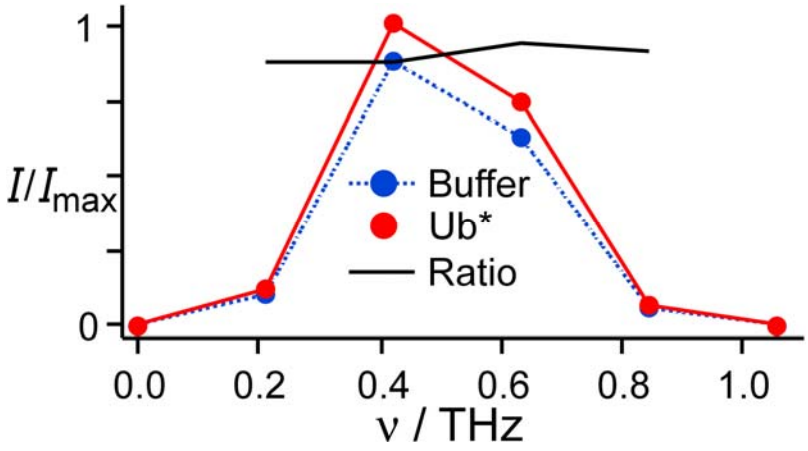


Figure 4.5 : Fourier Transform of transmitted Terahertz electric fields

It shows frequency spectrum of the Terahertz pulses used in the experiment, comparing transmission through buffer and through a 1.5 millimolar protein sample. The black curve shows that the absorption reduction caused by protein and hydration water is relatively constant over the frequency range.

4.4.2 Different slices of the Terahertz temporal pulse profile probe the same folding kinetics

Next, we used the change in transmitted Terahertz electric field between folded and unfolded protein solutions (typically 3-5%) to monitor folding kinetics. We mix

protein from a 6 M guanidinium hydrochloride buffer, where it is unfolded, to 0.86 M guanidinium hydrocholirde, where it folds. Refolding kinetics monitored by KITA provides a direct window on protein-solvent rearrangements during folding, such as the breaking of backbone-water hydrogen bonds and their replacement by backbone-backbone hydrogen bonds.

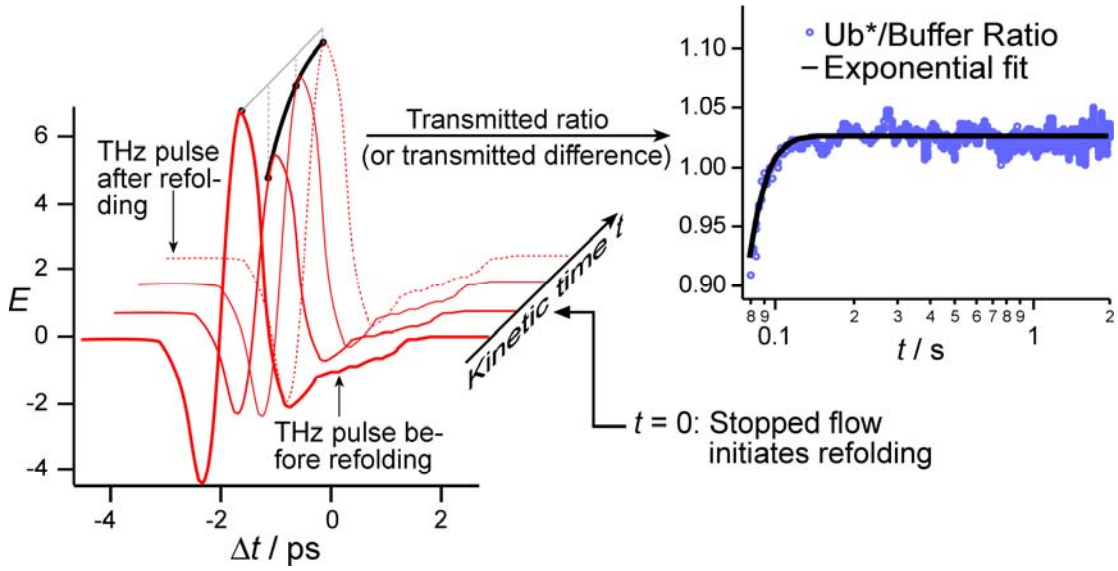


Figure 4.6 : KITA data collection scheme

Left: A succession of Terahertz pulses (red), each of ca. 4 picosecond duration (delay axis), passes through the sample for several seconds (kinetic time axis). At t=0, refolding of the protein is initiated, and the resulting Terahertz pulse attenuation and delay are monitored at a fixed delay time marked by black dots, resulting in a kinetic trace (blue). As the mixer is scanned in time *t* with respect to the Terahertz pulse, the field changes because the folded protein solution has different Terahertz absorbance and refractive index than the unfolded protein solution. This is repeated with buffer for reference. **Right:** Ratio of Ub*/buffer transmitted field at the pulse delay (0 picoseconds) of maximum field maps out folding kinetics at – 20 °C (blue). Only a fast single exponential is required to fit the data. The difference Ub*-buffer yields identical results to the ratio within measurement uncertainty.

[Figure 4.6] on the left shows schematically how KITA probes the evolution of collective protein-hydration water dynamics during refolding kinetics. Terahertz pulses probe the sample with delays between 0.05 to 5 seconds after the protein has been mixed into low guanidine hydrochloride buffer, tracing out refolding kinetics. These pulses are attenuated and delayed slightly differently because the sample has a different refractive index and absorbs differently as the protein undergoes folding. We collect a buffer sample for reference, and plot the kinetics either as the ratio of protein transmission/buffer transmission [Figure 4.6, right], or as the difference [Figure 4.4].

Because the change in absorption between protein solution and buffer is relatively small [Figure 4.4], both methods yield fits with the same time constant. The detectable kinetics are in the millisecond range under all conditions we measured.

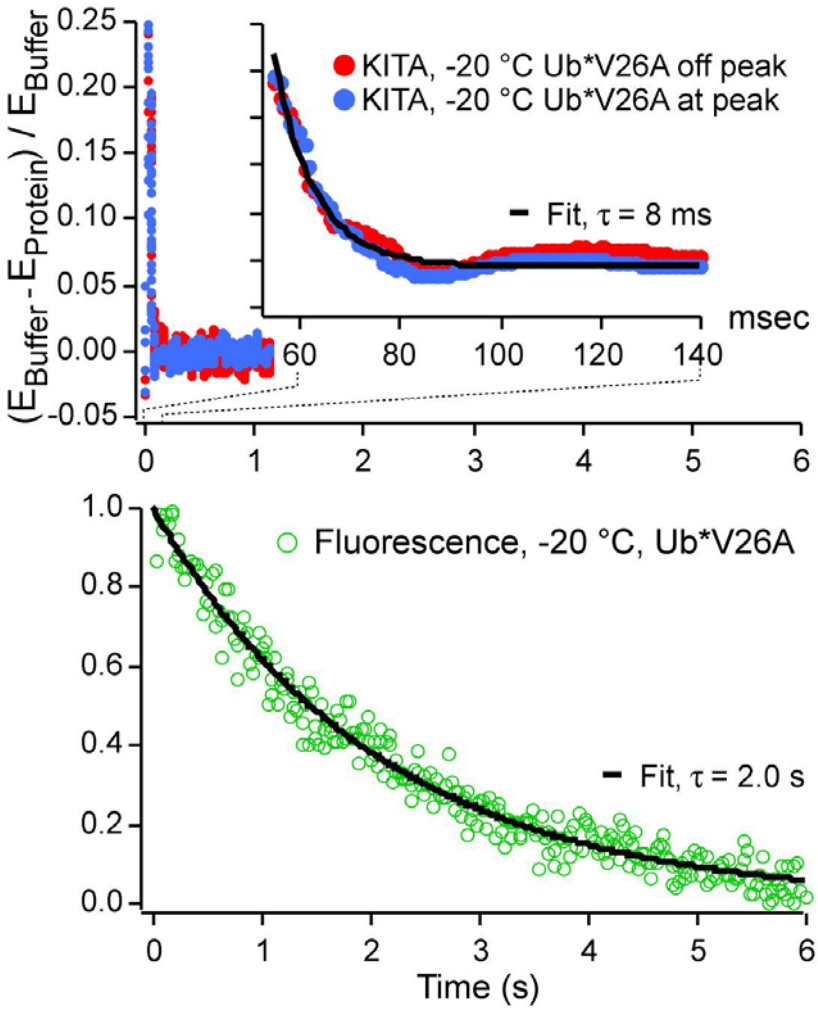


Figure 4.7 : Ub*V26A kinetics

Top: Terahertz transmission on and off the transmitted electric field peak yields identical millisecond kinetics at –20 °C. **Bottom**: Fluorescence-detected kinetics are much slower.

The nearly wavelength-independent absorption in [Figure 4.5] implies that the kinetics we detect should not depend on which part of the Terahertz pulse we probe, as indeed we find. The top panel in [Figure 4.7] shows the resulting kinetics for the Ub* V26A, detected at the peak and off the peak as indicated by the arrows in [Figure 4.4]. Within our measurement uncertainty, refolding kinetics are identical. Similar results were obtained for the other proteins we studied. Also described as [Figure 4.8], frequency

information contained at the peak and off-peak positions are consistently in the range of 0.2-0.8 Terahertz and one would expect KITA-detected kinetics not to be wavelength-sensitive in this range [Figure 4.5]. Therefore we can simply look at the kinetic trace from the peak of the electric field, where the signal-to-noise ratio is highest. For proteins whose absorbance varies markedly in the 0.2-0.8 Terahertz range, it would of course be interesting to detect the entire field instead of just plotting kinetics averaging over the 0.5±0.3 Terahertz range of the pulse from [Figure 4.5].

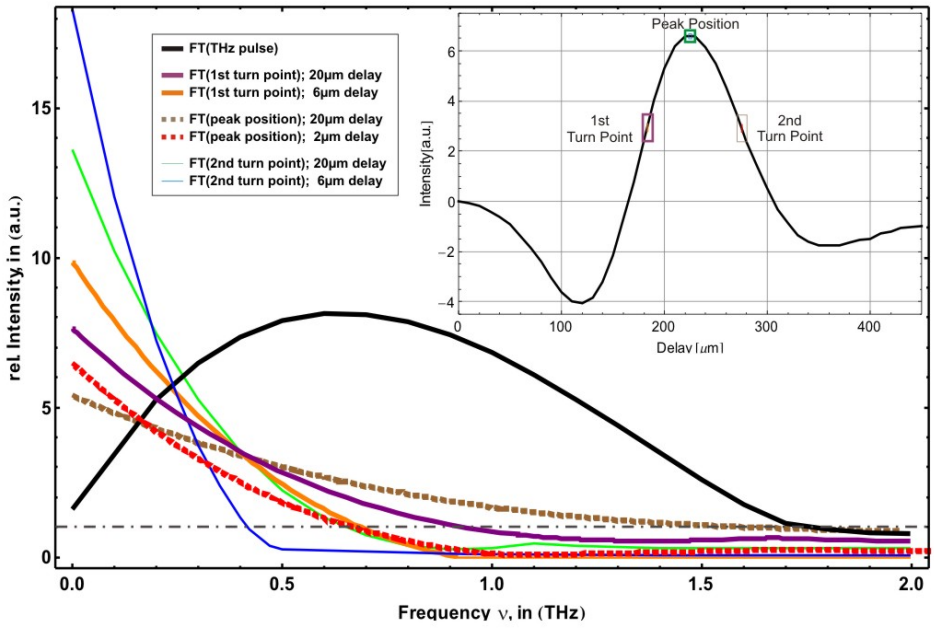


Figure 4.8 : Fourier Transform of Terahertz pulses at the peak and off-peak positions[8]

Fourier Transform was performed at the peak and off-peak positions (at 1st and 2nd turning point, in inset) with integration width of 6 µm and 20 µm.

4.4.3 KITA reaches equilibrium much faster than tryptophan fluorescence

The folding kinetics of Ub* and of its mutants turn out to be highly probe-dependent. In [Figure 4.7], the fitted KITA relaxation time of 8 ms is approximate due to

[8] By courtesy of Benjamin Born, Ruhr-University-Bochum, Germany

the 50 ms dead time of our stopped flow apparatus under the solvent conditions used. However, it is clear from the time scales in [Figure 4.7] that Terahertz transmission approaches the native equilibrium value of Ub* V26A nearly two orders of magnitude faster than tryptophan fluorescence.

We also carefully searched for changes in the Terahertz signal on the 1s time scale comparable to the fluorescence-detected kinetics. A typical result is shown for Ub* in the right panel of [Figure 4.6]. Within the signal-to-noise ratio of 5:1 to 20:1 achieved for the several Ub* mutants under various conditions, we were unable to observe any slow phase ([Figure 4.7], [Figure 4.9] and [Figure 4.10]). Thus for the current ubiquitin mutants KITA reports primarily on protein-solvent collective motions that equilibrate well before the tryptophan is packed into a native-like environment.

4.4.4 Four groups of observables emerge: KITA, fluorescence, CD and SAXS

We also compared KITA to CD- and SAXS-detected refolding kinetics of Ub* and Ub* I61A.[71, 97] To compare directly with the prior CD and SAXS experiments, we measured KITA under the same solvent conditions (40 mM phosphate buffer, 45% ethylene glycol in water buffer at pH 5.9). [Figure 4.9] and [Figure 4.10] compare KITA, fluorescence, CD (circular dichroism), and SAXS (small angle X-ray scattering) data for Ub* and Ub* I61A. KITA has only a fast millisecond phase. Fluorescence has only a slow phase. CD and SAXS show both phases. For both Ub* and Ub* I61A, the altered dynamics of hydration water and protein detected at 0.5 Terahertz go hand in hand with a rapid overshoot of the CD signal at 222 nm. The CD overshoot has been identified as due to formation of excess helical structure relative to the native state of ubiquitin, with the accompanying reduction of hydrogen bonds from the protein backbone to the hydration shell.[71, 83] At the same time, the CD signal has a slow response that matches the fluorescence data. This has been assigned to acquisition of native-like secondary structure after the protein has collapsed to a compact state. SAXS measurements indicate that Ub* and Ub* I61A indeed undergo a rapid collapse on the

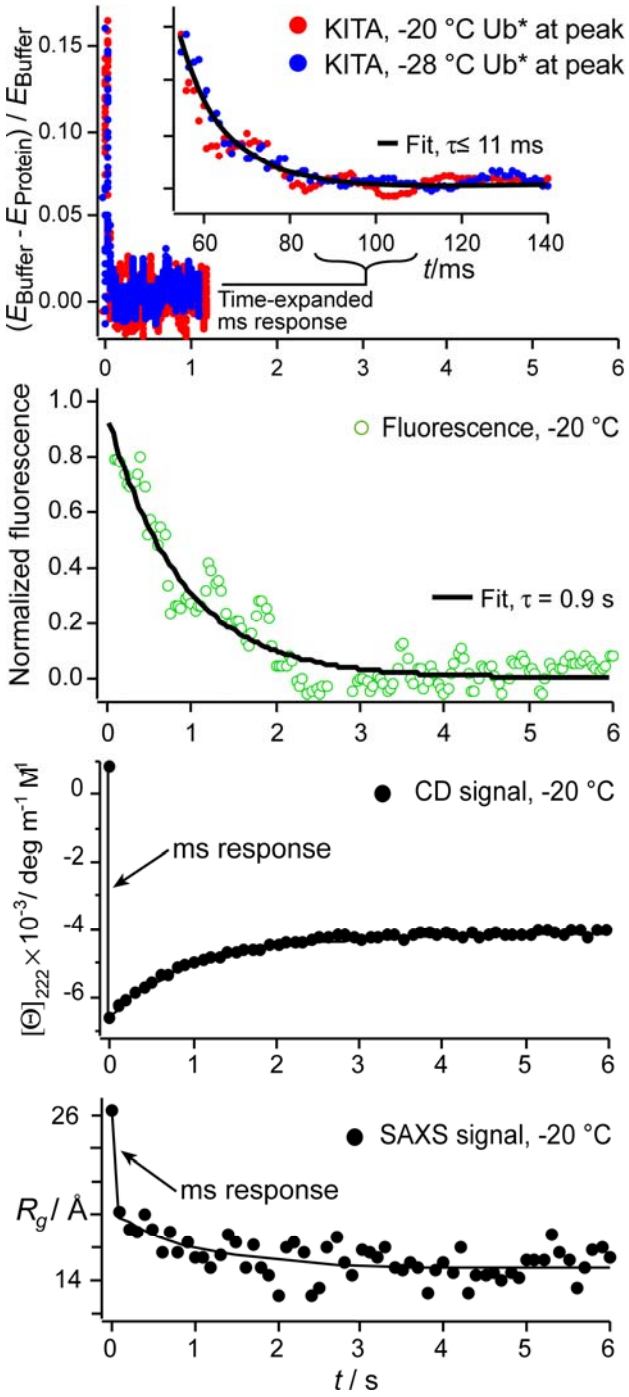


Figure 4.9 : Ub* pseudo-wild type (F45W) kinetics

KITA, fluorescence, CD, and SAXS refolding kinetics of Ub* (Due to the dead time, the KITA fit is an upper limit.) **Top:** the Terahertz-detected kinetics complete in ≈ 100 milliseconds and have weak or no temperature dependence between –20 and –28 °C in 45% ethylene glycol buffer. **Top center:** Tryptophan fluorescence-detected kinetics are an order of magnitude slower than Terahertz-detected kinetics. **Bottom center:** Circular dichroism-detected kinetics show both a ms phase that overshoots the native secondary structure content, and a slow phase that matches fluorescence and takes Ub* to the native state. **Bottom:** Small angle X-ray scattering also shows both phases: a millisecond contraction that matches the KITA signal, and further slow contraction to the native radius of gyration that matches the fluorescence signal. The bottom two panels are adapted from ref. [71].

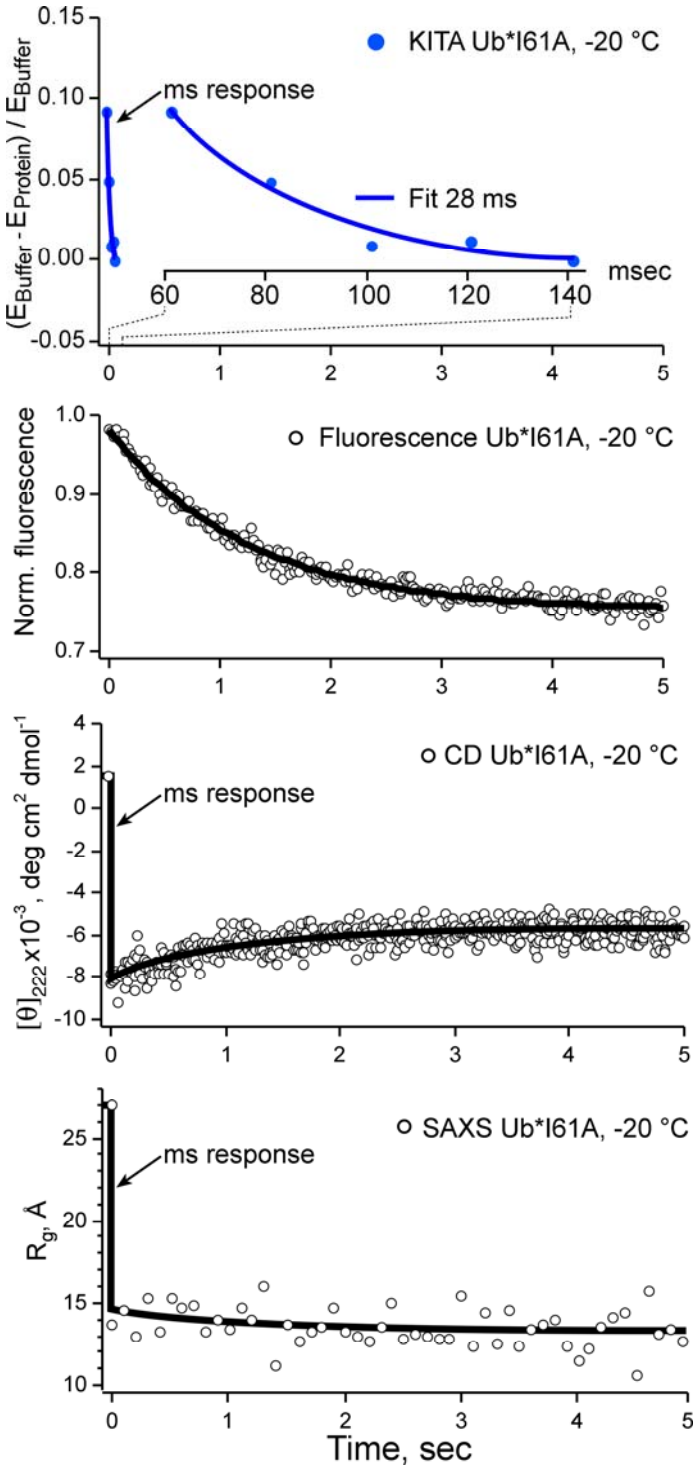


Figure 4.10 : Ub* I61A kinetics

Top: the Terahertz-detected kinetics are fast (<100 milliseconds). **Top center:** Tryptophan fluorescence changes much more slowly than Terahertz-detected kinetics. **Bottom center:** Circular dichroism-detected kinetics show both a millisecond phase that overshoots the native secondary structure content, and a slow phase that matches fluorescence and takes Ub* to the native state. **Bottom:** Small angle X-ray scattering of this mutant shows only a fast millisecond contraction to the native radius that matches the KITA time scale. The bottom two panels are adapted from ref. [71].

millisecond time scale, but like CD, SAXS data of Ub* also show a slow signature that matches the fluorescence. Thus the rearrangements of the solvent network detected by KITA occur during collapse and formation of early local secondary structure, whereas further rearrangements required by native packing are not picked up by KITA at 0.1-1 Terahertz.

4.4.5 Native protein flexibility has no systematic effect on early folding kinetics detected by KITA

Comparison of the KITA data for Ub*, Ub* V26A and Ub* I61A shows that native protein flexibility has no systematic effect on early Terahertz kinetics. We studied the folding kinetics of three mutants in addition to Ub*. Two of these mutants are single side chain truncations (Ub* V26A and Ub* I61A) of nonpolar residues. The valine is completely buried, while isoleucine 61 is largely buried.

As can be seen by comparing the top of [Figure 4.7], [Figure 4.9] and [Figure 4.10], Ub* I61A fits to a slightly slower exponential decay than Ub*, while Ub* V26A fits slightly faster. All are in the range of 18±10 ms. Considering the 50 ms dead time of the stopped flow, these differences are not significant. We also studied a double mutant truncating both positions (Ub* V26A I61V, data not shown), known to destabilize the native state by 20 kilojoules/mole.[97] This mutant has a single fast phase of 17±2 ms, again on the same time scale as [Figure 4.7], [Figure 4.9] and [Figure 4.10]. Thus the early kinetics detected by KITA are not strongly affected by mutations that destabilize the native hydrophobic core, even ones that significantly destabilize the native state. Interestingly, the same is true for the slow final stage of folding detected by fluorescence: the Val26Ala mutant in [Figure 4.7] is only a factor of two slower than the 14 kilojoule/mole more stable pseudo-wildtype Ub* shown in [Figure 4.9].[71]

4.4.6 Fast Ub* folding dynamics have no strong temperature dependence detected by KITA

[Figure 4.9] (top) compares the early Terahertz folding kinetics of Ub* at two different temperatures, –20 °C and -28 °C (chosen to allow direct comparison with existing SAXS data[71]). The traces fit to the same millisecond exponential decays within fitting uncertainty, and the same is the case for other mutants of Ub* (data not shown). We can use this to put limits on the activation energy for rearranging the hydration water network during early folding events, and find that the water network rearrangements and large amplitude protein motions probed by KITA have a very small activation energy, < 15 kilojoules/mole, whereas the later stage of folding monitored by fluorescence has a barrier of about 27.5 kJ/mole.

We proceed as follows. The rate is influenced by two factors: the activation energy $\Delta G^{\dagger}(T)$ controls how rapidly the protein can cross the barrier; the viscosity η controls how fast the protein chain can move in the solvent to get to the barrier. The resulting rate is given by Kramers [99] as

$$k = \nu^{\dagger}(\eta)\exp[-\Delta G^{\dagger}(T)/RT] \qquad (4.1)$$

Since ubiquitin is similar in size to cytochrome c, we can use the estimate of Eaton and coworkers, $\nu^{\dagger}(25\,^{\circ}C) \approx (1\ \mu s)^{-1}$, as a starting point for the prefactor.[100] For Ub*, the KITA signal precedes complete protein collapse to native-like compactness [Figure 4.9]. Thus the friction limiting the rate is largely solvent friction, not internal friction. If the solvent friction around the protein scales similarly with temperature as the bulk solvent friction (of course the absolute values, and even the scaling, could be different from bulk water), the corrected prefactor is

$$(1\ \mu s)^{-1}\,\eta(25\ ^{\circ}C)/\eta(T) \approx (19\ \mu s)^{-1} \text{ at } -28\ ^{\circ}C,$$
$$\text{and} \approx (13\ \mu s)^{-1} \text{ at } -20\ ^{\circ}C. \text{ [71]}$$

Combined with an observed upper limit of $(8\ \mathrm{ms})^{-1}$ on the rate coefficient k in [Figure 4.9], this yields a limit of $\Delta G^{\dagger} \leq 6RT$ = 15 kilojoules/mole for the activation free energy of Ub* as observed by KITA. The later folding stage monitored by fluorescence has a much larger barrier of about 11 *RT*, in line with barriers estimated for ubiquitin by other methods. [43]

For the Ub* I61V mutant in [Figure 4.10], complete collapse to the native radius of gyration is fast, so protein self-friction could contribute appreciably to the prefactor. However, it is very likely that the temperature scaling of self-friction is similar to that of the solvent, yielding a similar limit for that mutant. To the best of our knowledge, the temperature dependence of protein self-friction is not currently known from independent measurements. It is also worth noting that bulk viscosity scaling may be slightly weaker than η^{-1} (e.g. $\eta^{-0.7}$), but this will also have only a small effect on the limiting barrier over the temperature range discussed here.

4.5 Discussion

Monitoring the changes in terahertz absorption of a protein and of its hydration water by KITA during folding, coupled with site-directed mutagenesis, promises to provide a new experimental reaction coordinate that includes hydration water motion directly. As shown in our previous steady-state Terahertz measurements on ubiquitin,[77] hydration water makes a significant contribution to the difference between bulk solvent and millimolar protein solutions. This difference can be monitored as it relaxes from the denatured to the native value, as plotted in [Figure 4.7], [Figure 4.9] and [Figure 4.10].

The comparison of KITA and fluorescence shows that ubiquitin folds in at least two stages, the first of which is monitored by KITA and has a very low activation barrier $\leq 6\ RT$. The stage monitored by fluorescence has a much larger barrier of about 11 *RT*, in line with barriers estimated for ubiquitin by other methods.[43]

Conceptually, the interpretation of the Terahertz absorbance is straightforward. The dynamical hydration shell to which Terahertz absorption is sensitive has a thickness of 15-20 Å around proteins the size of ubiquitin.[19, 77] In the middle of the Terahertz band (2-3 Terahertz), addition of protein to bulk water increases absorbance over either protein or bulk water (by ≈10%), due to the strongly absorbing dynamical hydration shell. This "Terahertz excess" is taken away from the low Terahertz band (< 1 Terahertz), where the same protein solution absorbs 10-20% less than bulk water [Figure 4.5], creating a "Terahertz defect".[101] As shown in [Figure 4.7], [Figure 4.9] and [Figure 4.10], the Terahertz defect relaxes to its value in the native protein within less than 50 milliseconds after initiation of refolding.

For the 2-3 Terahertz data, we proposed a coupling of protein surface flexibility and hydration shell to explain the sensitivity of absorbance to side chain truncations in the core of the protein.[77] In contrast, kinetic measurements at 0.2-0.8 Terahertz are not sensitive to changes in protein flexibility that result from side chain truncations. The rates in [Figure 4.7], [Figure 4.9] and [Figure 4.10] are the same within experimental uncertainty (limited by the dead time), and show no systematic trend with native state flexibility.[66, 77] This indicates that a different mechanism influences the Terahertz spectrum at lower frequencies early during the folding process.

A comparison with circular dichroism data allows us to propose a tentative mechanism. The time scale observed by KITA is in line with the 6 millisecond upper limit set by circular dichroism spectroscopy on a fast phase that forms excess secondary structure, and with a 50 millisecond upper limit set by SAXS measurements that indicate complete or partial fast collapse of Ub* and Ub* I61V.[71, 97] Considering that during this time span, hydrogen bonds from the protein backbone to water are broken, and remade as intramolecular hydrogen bonds to form secondary structure, the agreement between circular dichroism and KITA is entirely plausible. We thus assign the KITA relaxation kinetics to formation of intermolecular hydrogen bonds early during protein folding.

If this interpretation is correct, investigation of KITA in deuterated water would be interesting. We predict that the Terahertz defect we observe at 0.5 Terahertz, and the excess we previously observed at 2.5 Terahertz, would both move to lower frequencies,

and would be very sensitive to the use of deuterated solvent. Likewise, monitoring the kinetics in water in the higher frequency 2-3 Terahertz band would be interesting. If this wavelength region is indeed more sensitive to surface flexibility differences induced in the native state by core mutations,[66] it should be able to pick up later stages (between 50 milliseconds and 2 seconds) where a native-like protein surface forms. Thus, the 2.5 Terahertz KITA signal could show a slow phase similar to the one observed by fluorescence, in addition or instead of the millisecond phase associated with a changing hydrogen bond network during secondary structure formation we have monitored here. Finally, we suggest the need for molecular dynamics simulations that compare the denatured and folded states of ubiquitin by computing the absorption at 0.5 Terahertz and 2.5 Terahertz from the dipole-dipole autocorrelation function,[63] and at the same time examining the protein-water and protein-protein hydrogen bonding.

The agreement between the circular dichroism (sensitive to protein backbone secondary structure) and KITA (sensitive to the protein-hydration water interaction) time scales shows how closely protein dynamics and solvent dynamics interact during folding. Although our measurements do not make a cause-effect distinction between protein and solvent dynamics, our results are in agreement with the hypothesis proposed by Frauenfelder and coworkers that some protein dynamics is slaved to solvent motions. The motions we observe by KITA would be the so-called alpha-fluctuations in the framework proposed by Frauenfelder.[47]

4.6 Acknowledgments

This work was supported by a grant from the Human Frontiers Science Program (MH and MG), by grant MCB-0613643 of the National Science Foundation (MG). BB wishes to thank the DAAD and the Ruhr-University Research School for financial support. We thank J. D. McDonald for the loan of a photomultiplier for the fluorescence experiments, and Matthias Krüger for programming the Terahertz data acquisition software.

Chapter 5 Development of the automated single molecule operating system (ASMOS) for a high throughput single molecule detector

5.1 Introduction

Most protein folding measurements have been conducted on the basis of bulk samples up to now.[102] What we get in a bulk is a statistical average of a protein ensemble. However, what if an individual protein behaves in a significantly different manner from the ensemble average? Bulk studies of protein folding are often frustrated by the presence of (either expected or unexpected) multiple species and multiple folding pathways, while a single molecule follows a single trajectory.[103] Since every single protein molecule might have different characteristics from the ensemble average, these differences can provide important information about the structure of the energy surface.

Single molecule spectroscopy is an important new approach for studying the intrinsically heterogeneous process of protein folding.[104] So far, several pioneering studies of a single protein molecule have been conducted by using mechanical force [103], single-molecule FRET [105], force-clamp atomic force microscopy [106], etc. The main difficulty with those experiments lies in the limited number of sampling due to the weak fluorescence from a single molecule, as well as the limited observation time with lengthy manual resetting gap between observations.

5.2 A high throughput single molecule detector

A new “high throughput single molecule detector” has been built for the study of protein folding energy landscapes on the basis of a single molecule, in collaboration with Krishnarjun Sarkar (a Ph.D. student in chemistry) and Dr. J. Douglas McDonald

(Professor of chemistry) under the supervision of Dr. Martin Gruebele. I developed the automated single molecule operating system (ASMOS), integrating the hardware controlling modules, fast data acquisition modules, and data analysis modules. Most of the instrument building works has been carried out by Krishnarjun Sarkar.

5.2.1 A lens cube assembly and 6 PMT tubes

We have designed a small (1 cm) lens cube, where the levitation of a 10 μm diameter droplet occurs by the Infrared laser guidance.[107-112] Every 10 μm diameter droplet, functioning as a "sample chamber", is generated by a custom made droplet generator on the top [113], under the piezoelectric control. [114] Diluted protein sample solution ascertains that each droplet contains one single protein only. As soon as two co-aligned infrared lasers trap a droplet, the excitation UV pulses focused into the single protein induce the excited states of a single protein, which enable the radiation of fluorescence photons. [Figure 5.1] shows a schematic of the lens cube assembly. This setup minimizes the extraneous interactions because there is no direct contact with any other materials inside the cube.

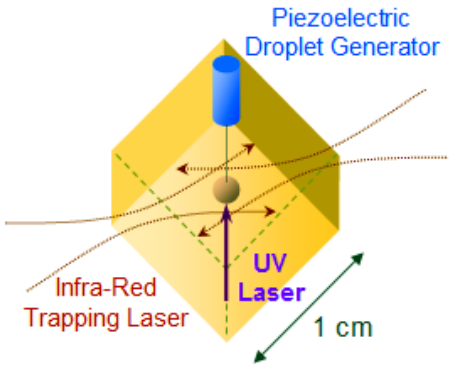


Figure 5.1 : A schematic of the lens cube assembly

A droplet is generated by a piezoelectric droplet generator from the top. It contains a single protein and is trapped by two IR laser beams at the center. The excitation UV laser pulses are focused into the protein for the fluorescence measurement. Both IR and UV beams comes into the cube through a tiny hole on the edge.

We built the lens cube assembly by putting together 6 pieces of the front lens, which is eventually mounted in front of each PMT tube. As shown in [Figure 5.2], all 6 PMT tubes are comprised of 2 lenses + 2 filters + 1 PMT components and attached to each surface of the lens cube assembly. This instrument was designed to collect as many as fluorescence photons selectively from all directions. Two cutoff filters (ET480/40m, Chroma) are installed for that purpose, and made of UV grade fused silica to maintain the minimum level of self-fluorescence. 6 PMTs (R7400 U-03, Hamamatsu) are installed for the fluorescence detection. This instrument makes possible the discrimination of polarization by the 4π steradian photon collection.

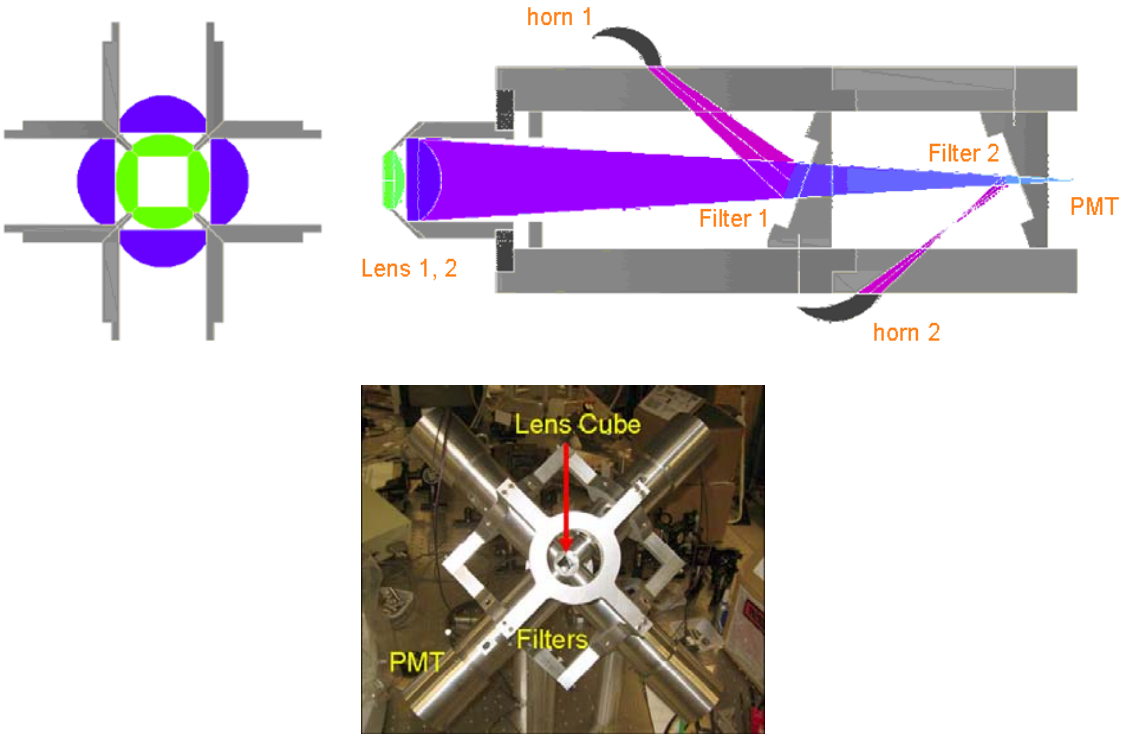


Figure 5.2 : The whole detection system with the lens cube assembly and 6 PMT tubes[9]

(**Left Top**) A cross section through four front pieces of the lens cube assembly. The region shaded in green is the first lens (Lambda research) and the one in blue is the second lens (R. Mathews optical works, Inc.). These lens sets are focused into each PMT. (**Right Top**) The schematic of a complete PMT tube made up of 2 lenses + 2 filters + 1 PMT. All the scattered UV probe beams are reflected by 2 filters and removed by absorption of the graphite coated horns (black). Only the fluorescence photons pass through the filters. This gives the very high signal to noise ratio required for a single molecule experiment. **(Bottom)** A photo of the lens cube + 6 PMT tubes held by the scaffold. The front PMT tube is removed to show the lens cube assembly (at the center) buried in the instrument.

[9]By courtesy of Krishnarjun Sarkar

5.2.2 Principles of operation for the droplet generation, laser guidance and measurement

A commercial controller box (JetDrive^{TM} III Controller, MicroFab Technologies) generates piezo-driving pulses in 50 Hz [115], at the command [116] of ASMOS (the automated single molecule operating system) through a serial port connection. Then a droplet of 10 μm diameter is generated and falls down from the tip of the piezoelectric droplet generator, synchronized with a piezo-driving pulse [Figure 5.3]. The initial ejection velocity of the droplet depends on the amplitude and shapes of the piezo-driving pulse. Eventually the droplet reaches a terminal velocity in few milliseconds and enters into the laser guiding region. (See [Chapter 6.5] for the simulation result.)

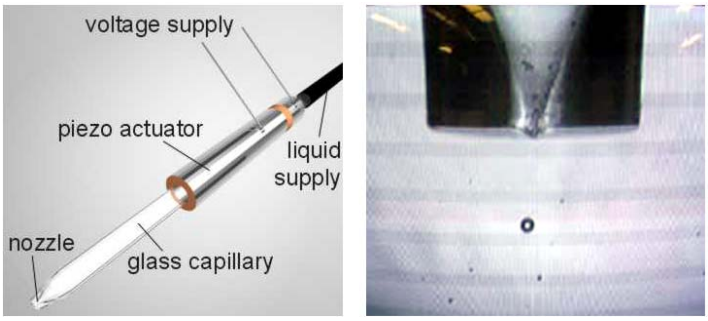


Figure 5.3 : The piezoelectric droplet generator[10]

On the right is the stroboscopic image of a drop just after coming out of the nozzle.

The use of a weakly convergent beam to first trap particles radially in the beam and subsequently guide them along the beam propagation axis has been termed laser guidance [117, 118]. For the precise and efficient laser guidance of a droplet we need to turn on two IR guiding diode lasers (ThorLabs) as soon as a droplet enters into the focused guiding region. The radius of the focused spot is about 50 microns. The 830 nm IR beam was chosen to minimize heat absorption by water and to avoid being detected by the PMT tubes.

A 5W green laser at 532nm (Millennia Pro, Spectra-Physics) pumps a Ti:Sapphire mode-locked laser (KMLabs), which generates 95 MHz pulses in 840 nm peak

[10] By courtesy of Krishnarjun Sarkar

wavelength with FWHM (full width at half maximum) of about 40 nm. This pulse beam goes into a custom-made Tripler, which produces 280 nm excitation UV pulses for the single molecule fluorescence measurements, and 420 nm reference pulses for the SYNC signal of the Data Acquisition Box (DAB). (See [Chapter 5.3.1] for the details of DAB.)

Currently, two different operating modes are available, one is "Continuous operating mode" and the other one is "On Demand operating mode". All procedures are fully automated in either mode.

In Continuous operating mode, two guiding diode lasers operate periodically in the same frequency as the generation of a droplet (50 Hz), but with a certain amount of delay time between the piezo-driving pulse and the guiding laser pulse [Figure 5.4]. The appropriate delay time is required for synchronization, since it takes (a certain) time until a droplet enters into the guiding region after a release from the nozzle. The main difficulty in Continuous operating mode lies in finding the appropriate delay time between the generation of a droplet and laser guidance.

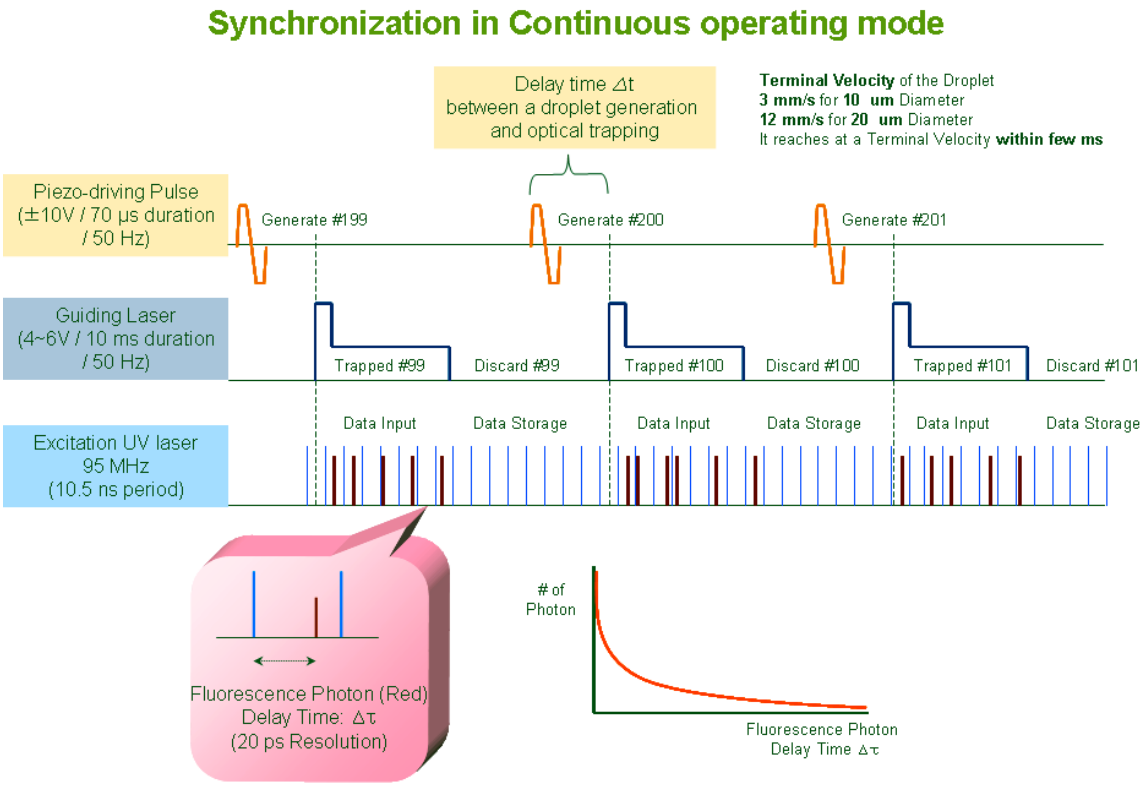


Figure 5.4 : Programming logic for the synchronization in Continuous operating mode

As soon as two co-aligned infrared lasers trap a droplet, the excitation UV pulse beam focused into the single protein will make possible the measurement of single molecule fluorescence. (See [Chapter 5.3] for the details)

Once 10 ms of the guiding time (or the exposure time) elapses, two guiding diode lasers are automatically turned off for the replacement of guided objects under the delicate control of the operating system. The automated sample replacement is very useful especially when the chromophores are used up. The droplet containing an old used-up protein starts to fall down and will eventually be evaporated and discarded. (We keep purging argon gas into the whole detection system, to remove unnecessary water molecules and maintain nearly zero humidity inside the lens cube assembly.) Then we are ready for a new sample, and repeat the same procedures in 50 Hz frequency for every single protein.

In "On Demand mode", the operation of two diode guiding lasers is triggered by the signal from a photodiode, which detects the UV scattering from the surface of a droplet. The signal from a photodiode guarantees that the protein is in the right position for the UV fluorescence measurement. Turning on the laser guidance by this signal will keep the protein in the UV focused region for enough time for the measurement. This event may take place in a non-periodic timely manner. For On Demand mode, we need an extra installation of a photodiode sensitive to the UV scattering inside the lens cube assembly.

In summary, all these techniques will provide the completely automated data acquisition and sample replacement, removing lengthy resetting times between observations. [Figure 5.5] shows the complete schematic of the high throughput single molecule detector we have been developing so far.

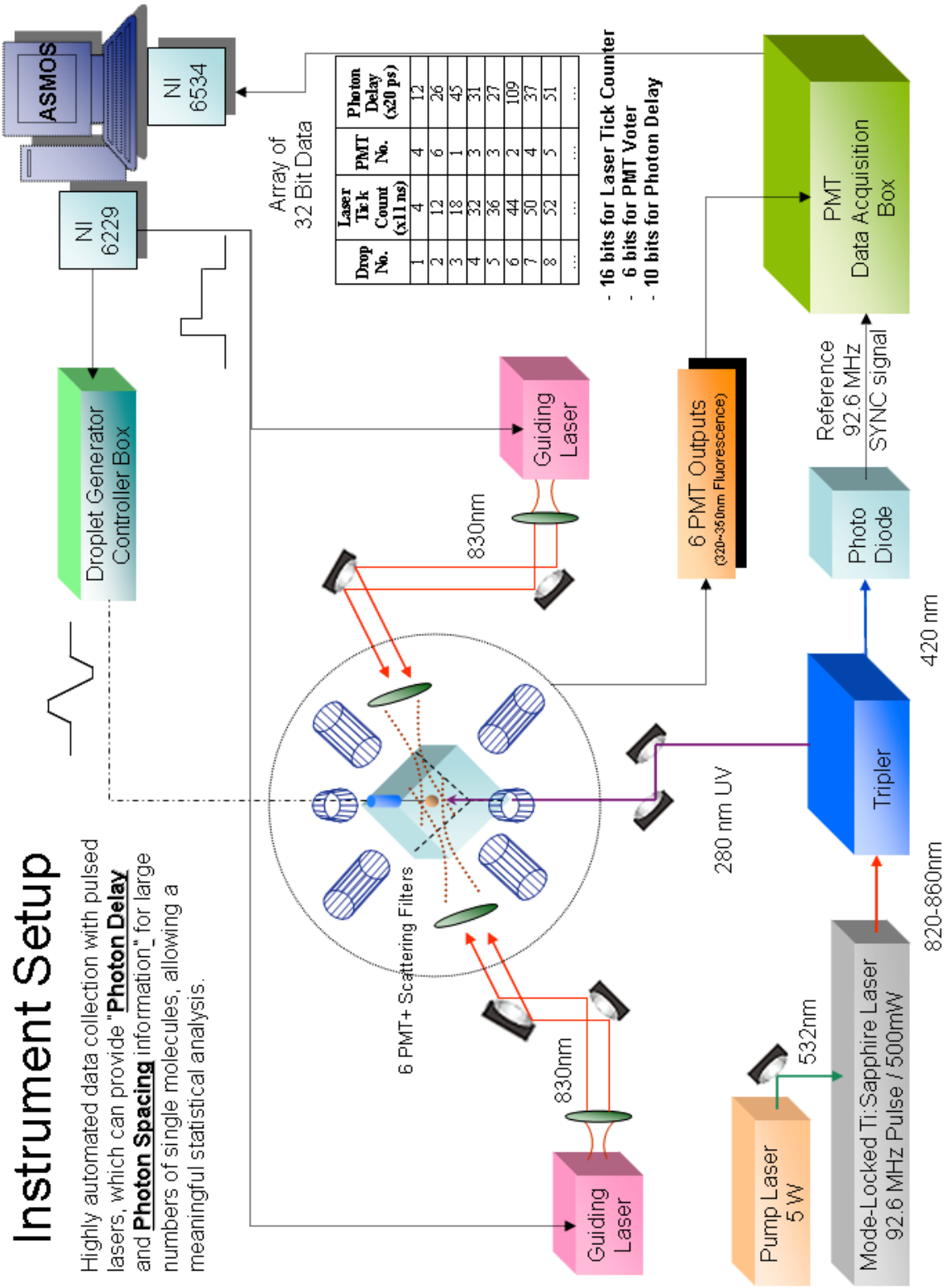


Figure 5.5 : The complete schematic of a high throughput single molecule detector

5.3 Hardware controlling module

5.3.1 Raw 32 bit binary data by Data Acquisition Box (DAB)

All single fluorescence photons detected by 6 PMT tubes are initially analyzed at the Data Acquisition Box (DAB). The custom made DAB was designed by Dr. Douglas McDonald and implemented by Mike Thompson of the SCS Electronic Services. As shown in [Figure 5.5] and [Table 5.1], DAB provides ASMOS with arrays of 32 bit binary data per a single photon, through the National Instruments (NI) PCI-6534 card device. (See [Chapter 5.3.2] for the NI devices.)

On the other hand, DAB has a RESET input port for the self-control. It stops functioning while the RESET input is being kept in HIGH state (=High voltage TTL signal is being applied to the RESET input). As soon as the RESET input switches into LOW state, DAB resumes functioning.

The data acquisition is fully synchronized with the 95 MHz excitation UV pulse train. DAB counts how many the reference UV pulses have passed for each photon input (photon arrival time) [high 16 bits], recognizes from which PMT the photon is coming [middle 6 bits], and measures the time gap between a fluorescence photon and the latest 95 MHz SYNC pulse [low 10 bits]. The 95 MHz SYNC pulses are split from a Tripler and 100% synchronized with the excitation UV pulses. Thus an efficient, time-correlated single photon counting setup is established.

The high 16 bits contains photon spacing information applicable to the analysis of time-correlation, while 6 bits in the middle represents polarization information of fluorescence. The fluorescence photon delay in the low 10 bits enables a statistical analysis of fluorescence photon decay and relaxation in the raw resolution of 10.3 ps.

In [Table 5.1], the raw 32 bit binary data (0000 0011 1111 0001 1000 0010 0010 0101) is sectioned out as high 16 bits, middle 6 bits, and low 10 bits. A conversion to a decimal number makes the interpretation of raw data easy. The high 16 bits (0000 0011 1111 0001) are converted to 1009 in a decimal number, while low 10 bits (10 0010 0101) are

converted to 549 in a decimal number. The middle 6 bits (1000 00) directly indicate that the photon is coming from PMT number 6. Therefore, this photon was detected after 1009 reference UV pulses since the start of the experiment by PMT number 6, and the fluorescence photon delay was 549 time units after the UV laser pulse #1009.

Table 5.1 : Interpretation of raw 32 bit binary data from DAB

Raw 32 bit binary data from DAB		High 16 bits, UV pulse Tick Count	Middle 6 bits, PMT #	Low 10 bits, Fluorescence Photon Delay
0000 0011 1111 0001 1000 0010 0010 0101		0000 0011 1111 0001	1000 00	10 0010 0101
Hexadecimal	03 F1 82 25	03 F1	20	225
Decimal	66159141	1009	32	549

It is worth noting that the UV pulse tick count in high 16 bits resets to zero (0000 0000 0000 0000) after it reaches its maximum value (1111 1111 1111 1111 in binary). The maximum UV pulse tick count is 65535 in a decimal number, so it continues to reset to zero per every 689.84 μs (= 65535 excitation UV pulses in 95 MHz). And the fluorescence photon delay in low 10 bits has a maximum value of 1024 (11 1111 1111 in binary). Since the fluorescence photon delay resets to zero when the next excitation UV pulse comes, the raw base time unit corresponds 10.3 ps. (= 10.5 ns period for 95 MHz UV pulse / 1024)

ASMOS is designed for the fast acquisition and massive storage of arrays of raw 32 bit binary data on a real time basis. (See [Chapter 5.4] for the details) Currently ASMOS is capable to manage massive photon data at the rate of up to 320 Mega Bits per second (= 40 Mega Bytes/s = 1,000,000 photon inputs per second) [11]. The maximum rate depends on the speed and bandwidth of the hard disk drives and CPU. This setup provides a much higher S/N ratio because of its high throughput.

[11] It is fully tested in a Microsoft Windows 2003 server with Intel Xeon 2.4 GHz CPU (quad-core Harpertown, 12MB L2 Cache) and a 10,000 rpm SATA II hard disk drive (Western Digital).

5.3.2 National Instruments devices

ASMOS interfaces with two National Instruments (NI) card devices via PCI (Peripheral Component Interconnect) bus for operating all instruments and acquiring the fluorescence photon data. NI PCI-6229 Multifunction Data Acquisition (DAQ) device generates analog outputs for the manipulation of the droplet generator controller box and the diode guiding lasers. NI PCI-6534 High-Speed Digital I/O device obtains raw 32 bit binary data from DAB, by performing digital pattern I/O. Both of NI devices are synchronized via RTSI (Real-Time System Integration) bus cable and connected to the instruments via SCB-68 I/O connector blocks. Here I present brief overviews of each NI device based on the official documents by National Instruments. [119-121]

NI PCI-6229 (multifunction device)

The National Instruments PCI-6229 is a multifunction M Series data acquisition (DAQ) board that incorporates advanced features as the followings to increase performance and accuracy.

- Four 16-bit analog outputs (833 kS/s)
- 48 digital I/O; 32-bit counters; digital triggering
- Correlated DIO (32 clocked lines, 1 MHz)
- NIST-traceable calibration certificate and more than 70 signal conditioning options
- Change detection

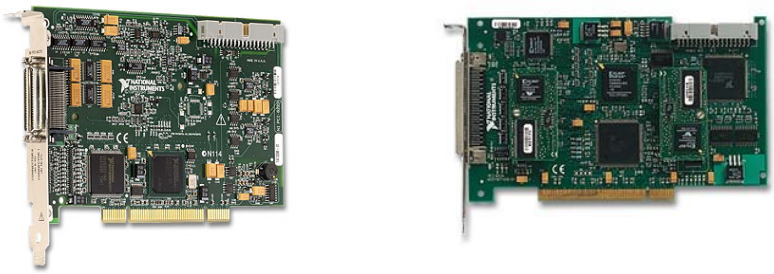


Figure 5.6 : NI PCI-6229 (left) and NI PCI-6534 (right)

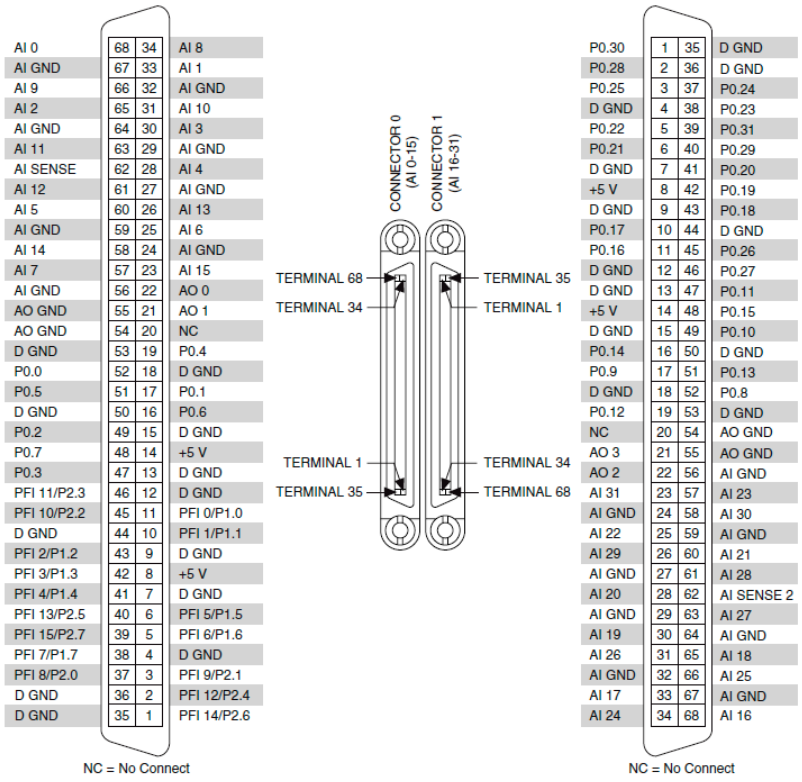


Figure 5.7 : Pin layout of NI PCI-6229

NI PCI-6534 (data acquisition device)

The National Instruments PCI-6534 is a high-speed, 32-bit, parallel digital I/O interface for PCI. The NI PCI-6534 performs pattern I/O and high-speed data transfer using a wide range of handshaking protocols at speeds up to 80 MB/s through onboard memory. It contains 64 MB of on board memory, which removes the dependency on the host computer bus for applications that require guaranteed transfer rates.[120] It features user-defined power-up states, start and stop triggering, pattern matching, and change detection.

We operate the 32 digital I/O lines as 32-bit ports for pattern I/O. The 32 digital I/O lines are physically connected to DAB for collecting raw 32 bit binary data. Initially NI PCI-6534 loads raw 32 bit binary data (patterns) into 64 MB of on board memory, and the patterns are transferred into the computer memory buffer continuously.

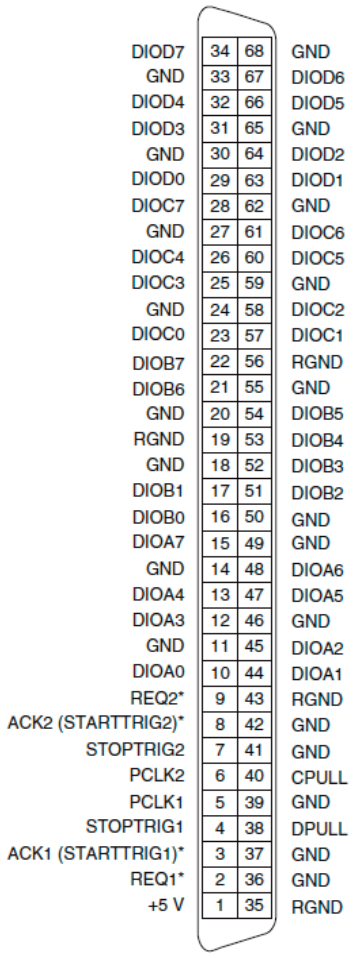


Figure 5.8 : Pin layout of NI PCI-6534

RTSI bus cable (synchronization)

RTSI stands for Real-Time System Integration. It is a bus found on many National Instruments devices that, when cabled together with a RTSI cable, is used to share and exchange timing and control signals between multiple boards. It is usually used for synchronization purposes. The RTSI bus cables are short, 34-conductor ribbon cables equipped with two to five connectors to link together a group of boards. The following figure shows an example of an extended five-board cable setup.

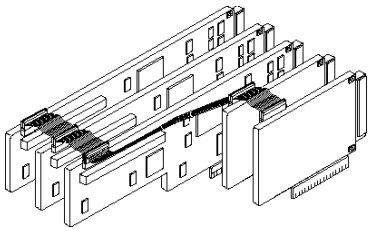


Figure 5.9 : RTSI bus cable

SCB-68 (connector block)

The SCB-68 is a shielded I/O connector block for interfacing I/O signals to plug-in DAQ devices with 68-pin connectors. Combined with the shielded cables, the SCB-68 provides rugged, very low-noise signal termination. Currently the SCB-68 is integrated as an essential part of DAB.

Figure 5.10 : SCB-68

5.3.3 Control of the droplet generator

The custom-made droplet generator operates with the 50 Hz piezo-driving pulses as typically shaped in [Figure 5.11]. The details described in this section are based on the official documents released by Microfab Technologies.[115, 116]

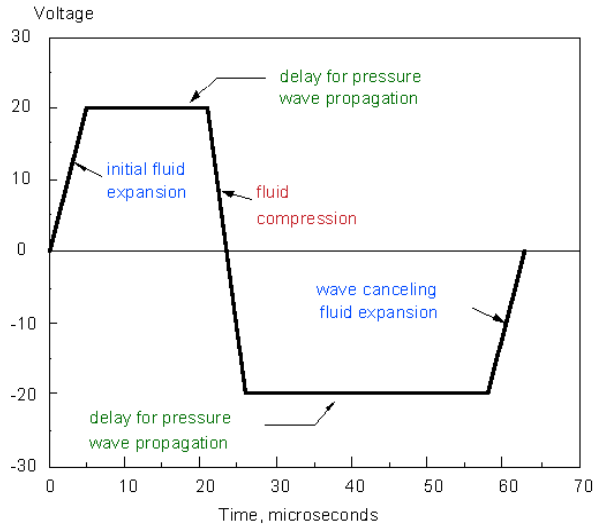


Figure 5.11 : A typical piezo-driving pulse for the droplet generation

From ref. [115]: MicroFab Technologies, *JetDrive™ III User's Guide*. 2003, MicroFab Technologies, Inc.: Plano, TX. p. 1-5.

Although the controller box (JetDrive™ III, Microfab Technologies) allows generating pulses within the voltage range from –140 V to 140 V, the typical piezo-driving pulse requires the pulse amplitudes at less than 40 V. The controller allows all three voltage levels (DC, positive pulse part, negative pulse part) to be adjusted in steps of 1 V, and all rise, dwell, and fall times in steps of 1 μs. The rise and fall times in most cases are around 3-5 μs, and the dwell times (durations of the positive and negative voltage pulse plateaus) are normally in the range 15-50 μs. A total pulse length can be extended up to 4095 μs and a longest single piece up to 3276 μs. The falling edge of the positive pulse excursion effectively determines the release time of a droplet from the droplet generator.

ASMOS sends user-defined parameter commands for customizing the piezo-driving pulses, to the controller box via a serial port communication (DTE to DCE; 9600 baud, 8 bits, no parity, and 1 stop bit). As an initialization process, commands are sent to the controller box one by one, and a response to each is returned back to ASMOS. The commands and responses are in a binary format. (See references [115, 116] for the details of the command set.)

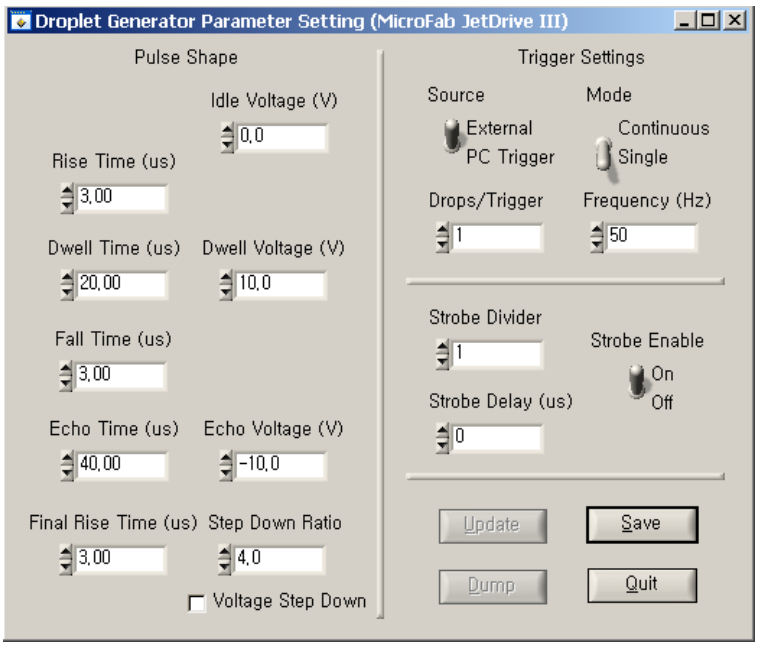


Figure 5.12 : Droplet generator parameter setting window in ASMOS

The "Pulse Shape" section on the screen allows adjustment of the pulse shape parameters. Editing of parameters occurs by directly typing a value or clicking up and down small icons on the left side.

The "Trigger Settings" section on the screen allows adjustment of triggering parameters. Editing follows the same pattern as described for the "Pulse Shape" command, except a "Source" and a "Mode" switch. In "Single" mode, the droplet is generated once per every signal trigger signal input.

In Continuous operating mode, soon after the controller box is initialized and ready for functioning, ASMOS gives an order for the NI PCI-6229 device to generate a periodic continuous pulse train as an external trigger source for the controller box. The rising edge of the external trigger pulse defines the execution timing of a piezo-driving pulse. The external trigger pulses are coming from one of the analog outputs in NI PCI-6229, and it makes sure that the piezo-driving pulse are fully synchronized with the laser guidance with a certain amount of delay time. [Figure 5.4]

It is rather simple for the case of On Demand operating mode, since

synchronization is not required. When the controller box is initialized and ready for functioning, ASMOS just sends a command to the controller box directly for the generation of the piezo-driving pulse. The controller box uses an internal trigger for the continuous periodic pulse generation.

5.3.4 Control of the guiding lasers

The power of the IR guiding laser is proportional to the voltage applied to the diode laser. ASMOS gives an order for the NI PCI-6229 device to generate the user-defined analog guide signals for customizing the IR guiding lasers. The analog guide signals are coming from two analog outputs in NI PCI-6229 and going into each diode laser. A typical shape of the analog guide signal is shown in [Figure 5.13].

Figure 5.13 : A typical analog guide signal for the guiding lasers

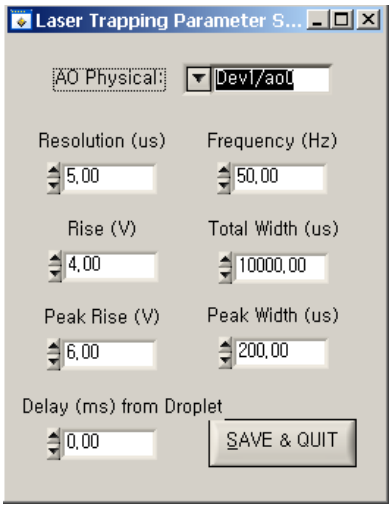


Figure 5.14 : Laser trapping parameter setting window in ASMOS

For a short time of 200 μs, we apply a higher 6 V to generate the high power guiding laser, positioning a droplet at the focus of excitation UV pulses. Then we switch to a lower 4 V to keep the droplet for the fluorescence measurement for 10 ms. The execution timing of the guiding lasers is determined by the current setting of the operating mode, as defined in [Chapter 5.2.2] and shown in [Figure 5.4].

5.3.5 Hardware alignment

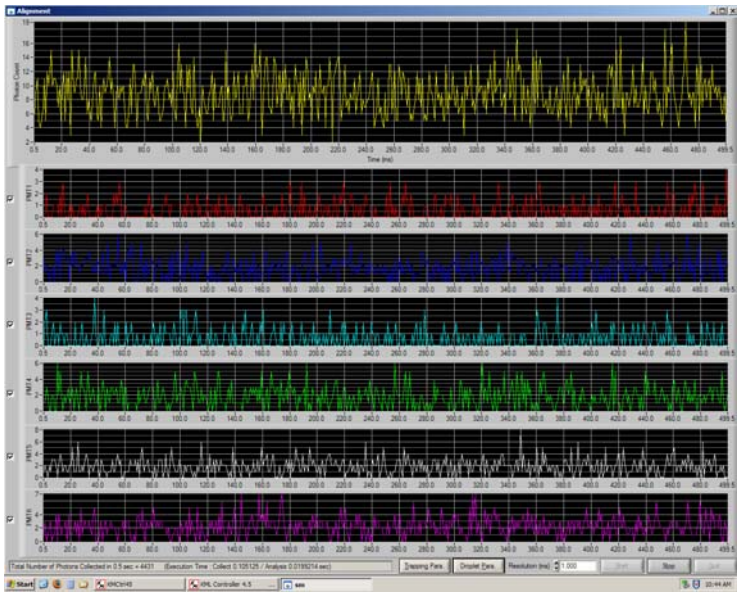


Figure 5.15 : Alignment window in ASMOS

In order to reach a maximum signal-to-noise ratio, every instrument must be aligned properly, and especially the alignment of the UV excitation laser and the droplet generator are most essential parts in ASMOS. For user's sake, one can use an alignment module to tweak hardware in a real time base. The alignment window can start by clicking the "Alignment" button on the main panel of ASMOS. During the hardware alignment, the user can monitor how photon spacing and count information changes in a real time base, typically automatically updated every 0.6 sec (=reading time on the main panel + 0.1 sec). This module displays the total photon count on the top, and photon counts by each PMT below.

5.4 Fast data acquisition module with multiple threading

The entire ASMOS source codes are written in C language, in the National Instruments LabWindows/CVI 8.5 development environment. ASMOS is installed in a Microsoft Windows 2003 server with Intel Xeon 2.4 GHz CPU (quad-core Harpertown, 12MB L2 Cache) and SATA II RAID systems (mode 0) for massive storage. In ASMOS source codes, the NI PCI-6229 (multifunction device) is defined as "Dev 1", and the NI PCI-6534 (data acquisition) as "Dev 2". Refer to [Appendix C] for the source codes and manuals.

5.4.1 Main panel - hardware initialization

First of all, all NI devices must be reset appropriately. [DAQmxResetDevice();] ASMOS loads all user parameter inputs [Initialize_Parameter(); InitializeMicroJet();] and reads the calibration profiles for DAB and PMT. [TimeCalibration(); build_corrections();] Then ASMOS is waiting for a user input in a stand-by mode, unless the user clicks any buttons on the main panel.[RunUserInterface();]

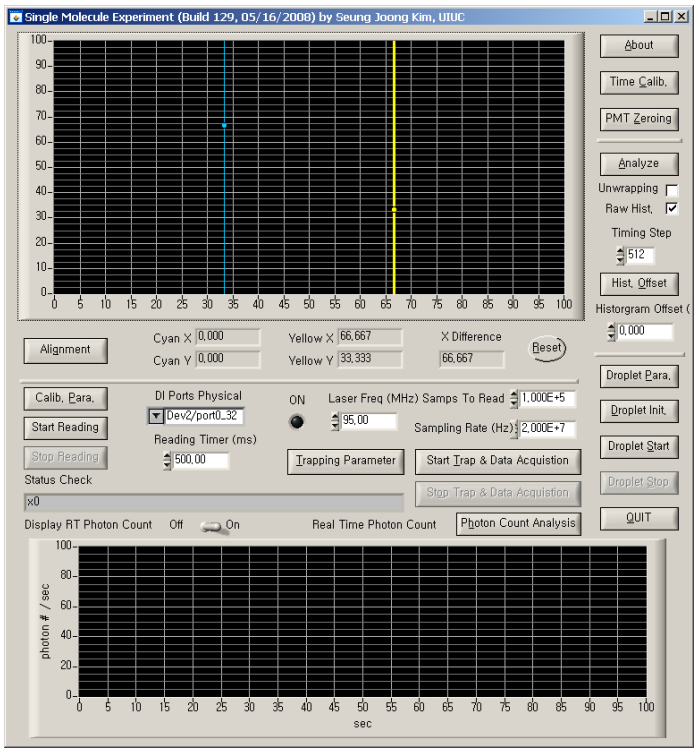


Figure 5.16 : The main panel of ASMOS

Most of sub-windows and applications can start by clicking buttons on the main panel

Next, the user needs to initialize the droplet generator by clicking “Droplet Init.” button. [SerialPort_Connection(); Initial_Connection();] In order to start the droplet generator, just click “Droplet Start” button on the main panel. [Droplet_Start();] To stop operating the droplet generator, click “Droplet Stop” button. [Droplet_Stop();]

The laser guidance and data acquisition begins when the user clicks “Start Trap & Data Acquisition” button on the main panel. [StartTrapDAQ_Thread();] ASMOS configures all hardware parameters according to the current setting of the operating mode.

In Continuous operating mode,

1. The PFI 0 port (P1.0) in NI PCI-6229 is assigned as a TRIGGER INPUT for the data acquisition.

2. Two analog output ports (AO0, AO1) in NI PCI-6229 are set up for the laser guidance. Two analog signals are generated periodically in 50 Hz.

3. The PFI 12 port in NI PCI-6229 (ctr0 out: Dev1/PFI12) is set up for a pseudo RESET signal and internally wired into the PFI 4 port (Dev2/PFI4) in NI PCI-6534, which is going to be functioning as the real RESET input to DAB. The RESET signals are generated periodically in 50 Hz.

4. The PFI 13 port in NI PCI-6229 (ctr1 out: Dev1/PFI13) is set up for an EXTERNAL TRIGGER signal for the droplet generator. The EXTERNAL TRIGGER signals are generated periodically in 50 Hz.

All three signal outputs (the guiding laser pulse, the RESET signal into DAB, and the EXTERNAL TRIGGER for a droplet generator) are triggered by the TRIGGER INPUT signal at the PFI 0 port (P1.0) in NI PCI-6229 and synchronized. In the sources codes, another name for the TRIGGER INPUT is “ao/StartTrigger”.

In On Demand operating mode,

We need a photodiode signal which detects the UV scattering from the lens cube assembly. The PFI 4 port in NI PCI-6229 ("/Dev1/PFI4") is reserved to be connected with the photodiode. We use the "retriggered pattern I/O" for the operation.

1. The (P0.7) port in NI PCI-6229 is assigned as a pseudo RESET signal, and initially set HIGH. It must be physically wired into the PFI 4 port (Dev2/PFI4) in NI PCI-6534, which is going to be functioning as the real RESET input for DAB.

2. We generate the "Retriggered External Timing Source" (RETS) for the laser guidance and the RESET signal to DAB. Counter output function of NI PCI-6229 is used for this purpose. RETS is coming out from the counter1 output in NI PCI-6229 (Ctr1 out: Dev1/PFI13), triggered by the counter1 gate input in NI PCI-6229 (Ctr1 gate: Dev1/PFI4), which is physically connected to the photodiode input for the scattering photon detection.

3. Two analog output ports (AO2, AO3) in NI PCI-6229 are set up for the laser guidance by using RETS (=Ctrl1 Internal Output) as a sample clock timing source.

4. We generate the On Demand pseudo RESET Signal at the (P0.7) port, by using RETS (=Ctrl1 Internal Output) as a sample clock timing source. It is already physically wired into the PFI 4 port (Dev2/PFI4) in NI PCI-6534, which is going to be functioning as the real RESET input for DAB.

The laser guidance and RESET signal are synchronized each other since they share the same sample clock timing source, which is the RETS. The droplet generator is triggered by an internal trigger from the controller box.

5.4.2 Change detection of the RESET signal and multiple threading

Change detection is defined as a transition on one or more (digital) input lines that causes the entire group to be captured in hardware. With change detection, we can automatically trigger a certain operation upon a digital change of state. ASMOS uses change detection on the (P0.0) input in NI PCI-6229. Because the (P0.0) input is externally (by wire) connected from the RESET outputs (PFI 12 in Continuous operating mode and (P0.7) in On Demand operating mode), the change detection allows detecting the change (the rising edge) of the RESET signal and creates a Windows event, with a resolution of 150 ns. As soon as ASMOS captures this event, it executes a data reading thread [DAQ_thread();], which delivers all raw data in the computer memory buffer [Chapter 5.3.2] into ASMOS. Since DAB stops functioning when the RESET input is in HIGH state, the detection of the rising edge of the RESET ensures that DAB doesn't collect any raw data when ASMOS performs the data reading or data storage process.

Eventually a data reading thread flushes all raw data into a Thread Safety Queue (TSQ) for the protection of raw data shared by multiple threads. ASMOS uses multiple threads for improved performance and enhanced security during data acquisition and storage. Here I present quotes from the National Instruments website about the multi-threading.[121]

"With *multithreading*, applications can separate their own tasks into individual threads. In a multithreaded program, the OS directs each thread to execute code for a period of time, referred to as a time slice, before switching execution to another thread. The act of stopping execution of one thread and starting execution of another is referred to as a thread switch. The OS typically can perform thread switches quickly enough to give the appearance of concurrent execution of more than one thread at a time.

...

The most common reason is to separate multiple tasks, one or more of which is time-critical and might be subject to interference by the execution of the other tasks. For example, a program that performs data acquisition and displays a user interface is a good

candidate for multithreading. In this type of program, the data acquisition is the time-critical task that might be subject to interference by the user interface task. While using a single-threaded approach in a LabWindows/CVI program, you might decide to pull data from the data acquisition buffer, plot the data to a user interface graph, and then process events to allow the user interface to update. If the user chooses to operate your user interface (for example, by dragging a cursor on a graph), the thread continues to process the user interface events and does not return to the data acquisition task before the data acquisition buffer overflows. Using a multithreaded approach in a LabWindows/CVI program, you might put the data acquisition operations in one thread and display the user interface in another thread. This way, while the user is operating the user interface, the OS performs thread switches to give the data acquisition thread time to perform its task."

5.4.3 Data acquisition by performing pattern I/O with the NI PCI-6534

With pattern I/O, we can acquire raw 32 bit data (patterns) under timing control of a REQ clock signal input. We acquire raw data (patterns) on every rising edge of a REQ clock signal, which are generated by DAB and received through the REQ port (PFI 2) in NI PCI-6534. The low time and high time of the REQ signal must each be >20 ns. The minimum duration for a period of the REQ signal is 50 ns. Refer to the reference [120] for the details of pattern I/O.

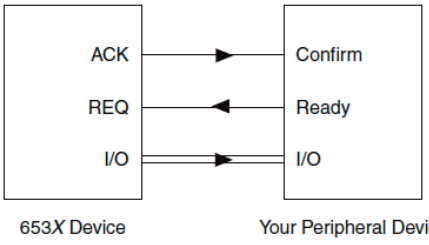


Figure 5.17 : Connecting signals in Pattern I/O (from ref. [120])

As shown in [Figure 5.18], the NI PCI-6534 device loads raw 32 bit binary data (pattern I/O) from DAB into 64 MB of on board memory, and the patterns are transferred

into the computer memory buffer continuously. The data reading thread delivers all data in the computer memory buffer into ASMOS, and eventually into TSQ for the storage.

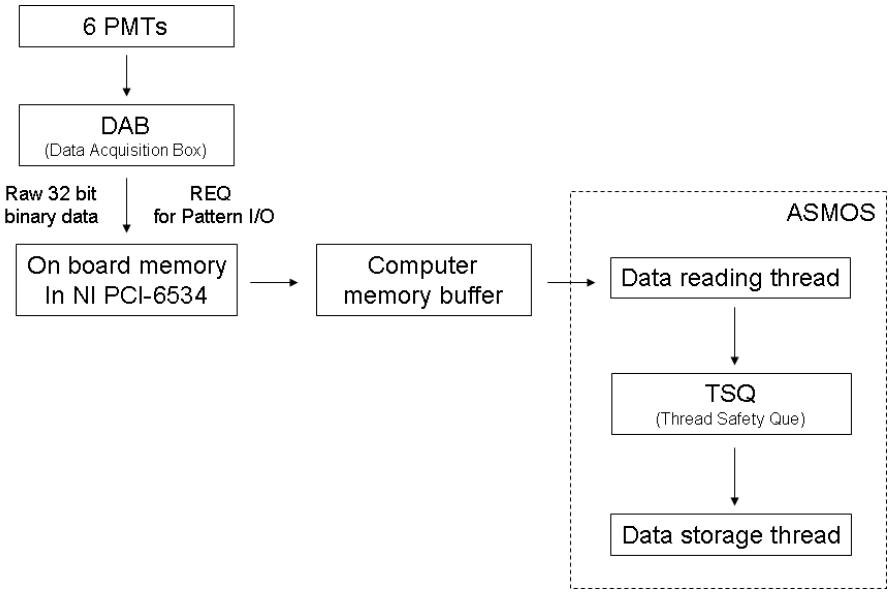


Figure 5.18 : A flow chart for the raw data processing in ASMOS

5.4.4 Thread safety queue (TSQ) and the critical section (CS)

We introduce a Thread Safety Queue (TSQ) to protect all raw data acquired and shared by multiple threads. The raw 32 bit binary data are sent to TSQ for the storage and real time analysis, and TSQ guarantees the safety and reliability of all raw data shared in the multiple threading environments.

According to the National Instruments website, we can safely pass data between threads with TSQ. It is most useful when one thread acquires the data and another thread processes that data. TSQ handles all the data locking internally. Generally, a secondary thread in the application acquires the data while the main thread reads the data when it is available and then analyzes and/or displays the data.[121]

In addition, we use a critical section (CS) for the exclusive thread running. CS

guarantees that only one of the same kinds of threads will be executing at the moment, while the other threads are just waiting for their turn in a stand by mode. It is a "thread lock", in a sense that only one of the same kinds has the key to open a lock.

CS is very useful and efficient tool for managing multiple threads, especially in case of huge numbers of raw data far more than the capacity of ASMOS. The execution of the data reading and data storage threads can be delayed or even suspended in this case. CS determines the execution order of the multiple threads competing to each other.

5.4.5 Massive data storage by storage threads and overlapped I/O

I have built 4 TB (terabyte) of a massive storage system with SATA II RAID (mode 0). Finally ASMOS executes a data storage thread [Storage_thread();], which stores raw 32 bit data in TSQ into the RAID system and plots a real-time histogram of the fluorescence decay per every 0.5 sec.

Currently ASMOS can store massive data sets at the rate of up to 100 Mega Bytes per second[12]. The Overlapped I/O operation is set up at the data storage thread for better performance. The Overlapped I/O is an asynchronous I/O operation on files. Microsoft MSDN library describes the principle of Overlapped I/O as the followings.

"When a function is executed synchronously, it does not return until the operation has been completed. This means that the execution of the calling thread can be blocked for an indefinite period while it waits for a time-consuming operation to finish. Functions called for overlapped operation can return immediately, even though the operation has not been completed. This enables a time-consuming I/O operation to be executed in the background while the calling thread is free to perform other tasks. For example, a single thread can perform simultaneous I/O operations on different handles, or even simultaneous read and write operations on the same handle." [122]

[12] Benchmarked by SiSoft Sandra software, United Kingtom

5.5 Data analysis module for a single protein molecule

The data analysis module has been developed for the two different user interfaces. The GUI (graphical user interface) version is integrated with AMOS in NI Labwindows/CVI 8.5 environment for a Windows machine. The universal ANSI C version is an independent program which can work regardless to the operating system. It loads user parameter inputs by a configuration file (*.ini). Both versions give the same result.

5.5.1 Calibration of the instruments by using a uniform white light source

DAB gives us useful information about the raw photon delay time between a fluorescence photon and the latest excitation UV pulse. It is encoded as low 10 bits in a raw 32 bit binary data. But this raw photon delay time doesn't have an actual physical meaning unless it is adequately corrected by calibration of all instruments. All National Instruments devices have self-calibration functions, so we don't need to worry about them in a normal working condition. But DAB is custom made and it is recommended to calibrate it regularly for the highest accuracy.

Calibration of the whole detection system (DAB and PMT) is performed by replacing the excitation UV pulses with a uniform white light source. It consists of random timing photons from all directions in all visible wavelengths. We still need a 95 MHz SYNC pulse train for the reference input. Calibration of the detection system is focused onto the photon delay time in low 10 bits. By calculating a histogram of the photon delay time, we can analyze the real response characteristic of the detection system. Dr. McDonald developed an algorithm converting the raw data to the uniformly corrected one by generating a conversion table for the white light source. This algorithm is integrated to ASMOS. Once the user calibrates the detection system, ASMOS keeps the calibration settings for the future usage.

For a long time (> 30 minutes) data acquisition, the auto-calibration module has been developed for user convenience. It automatically calibrates the whole detection system according to user input parameters. One can launch the parameter setting window

by clicking the “Calib. Para.” button on the main panel.

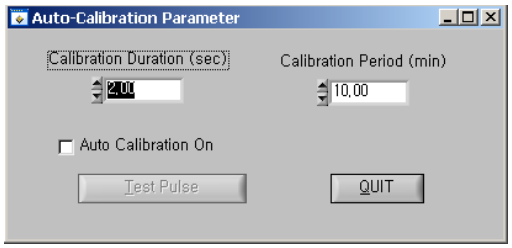


Figure 5.19 : Auto-calibration parameter setting window in ASMOS

5.5.2 PMT zeroing

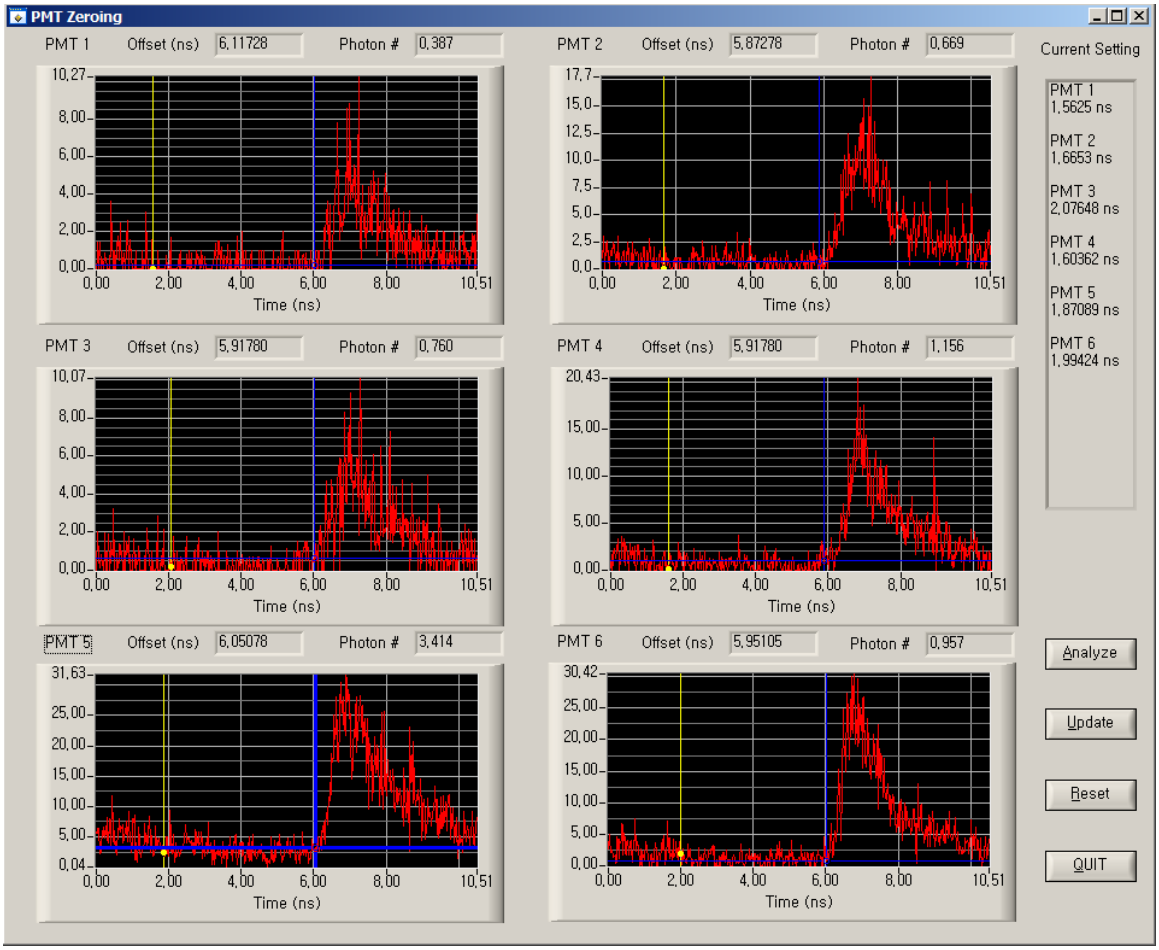


Figure 5.20 : PMT zeroing window in ASMOS

The start time of a fluorescence photon delay may be different from each PMT as shown in [Figure 5.20]. It is due to the different response of each timing circuit in DAB. In this case, the user can adjust PMT delay timings by setting zero points for each PMT, manually by dragging the cursors.

5.5.3 Analysis

The analysis module reads a binary file containing arrays of raw 32 bit binary data or a text file that is already analyzed before. The output file format of analyzed file is ASCII so that it is easily readable in any computer system.

For the analysis of a binary file, the analysis module sections out the raw 32 bit binary data as high 16 bits, middle 6 bits, and low 10 bits as described in [Chapter 5.3.1] and [Table 5.1]. It generates ASCII text files (*.txt) which consists of four columns. The first column shows how many the reference UV pulses have passed for each photon input and denotes the actual photon arrival time. It is used for the time-correlation analysis later. The second column shows the index number of PMT, and the third column presents the raw photon delay time between a fluorescence photon and the latest excitation UV pulse. The last column has the corrected photon delay time and contains a real physical meaning of the fluorescence decay. It is used to generate a histogram file for the fluorescence lifetime analysis. The GUI version of ASMOS automatically plots the histogram of the fluorescence decay for a single protein molecule, in the main panel.

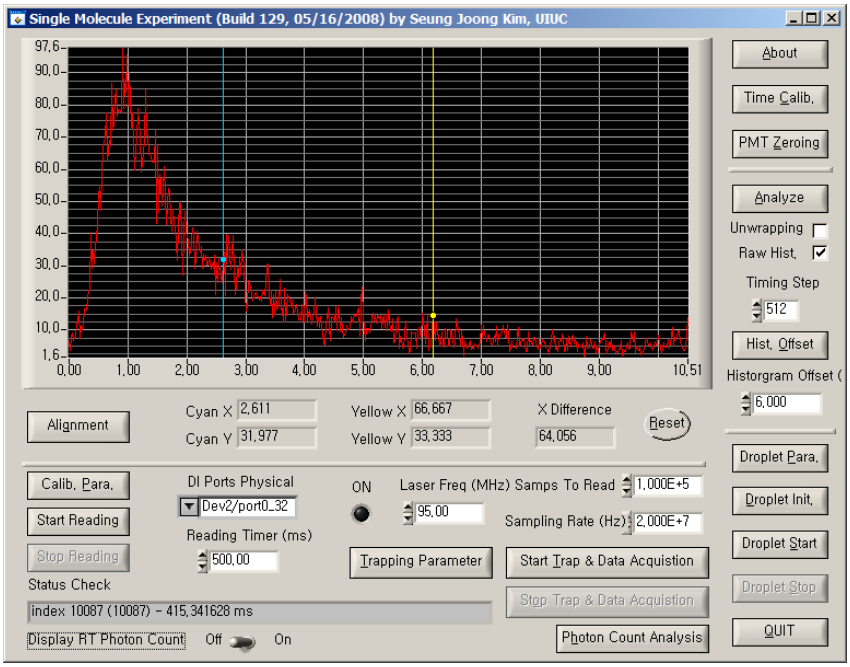


Figure 5.21 : Histogram of the fluorescence decay for a single protein molecule

For the re-analysis of a text file, the analysis module simply copies the contents in the first, second, and third column into a new text analyzed file. But it generates a new corrected photon delay time between a fluorescence photon and the latest excitation UV pulse. This is useful when the user has a new calibration file for the instruments.

For photon spacing (counting) information used for the correlation analysis, a "Photon Count Analysis" module has been developed. One can start the module by clicking "Photon Count Analysis" button on the main panel of ASMOS. It reads the raw binary or pre-analyzed ASCII file as a user input, and plots the photon spacing information in a real time coordinate. It also shows photon spacing information collected by each PMT, which can be used for the future analysis of fluorescence polarization. One can easily drag the cursors to get the coordinates and select the ranges for plotting and saving data.

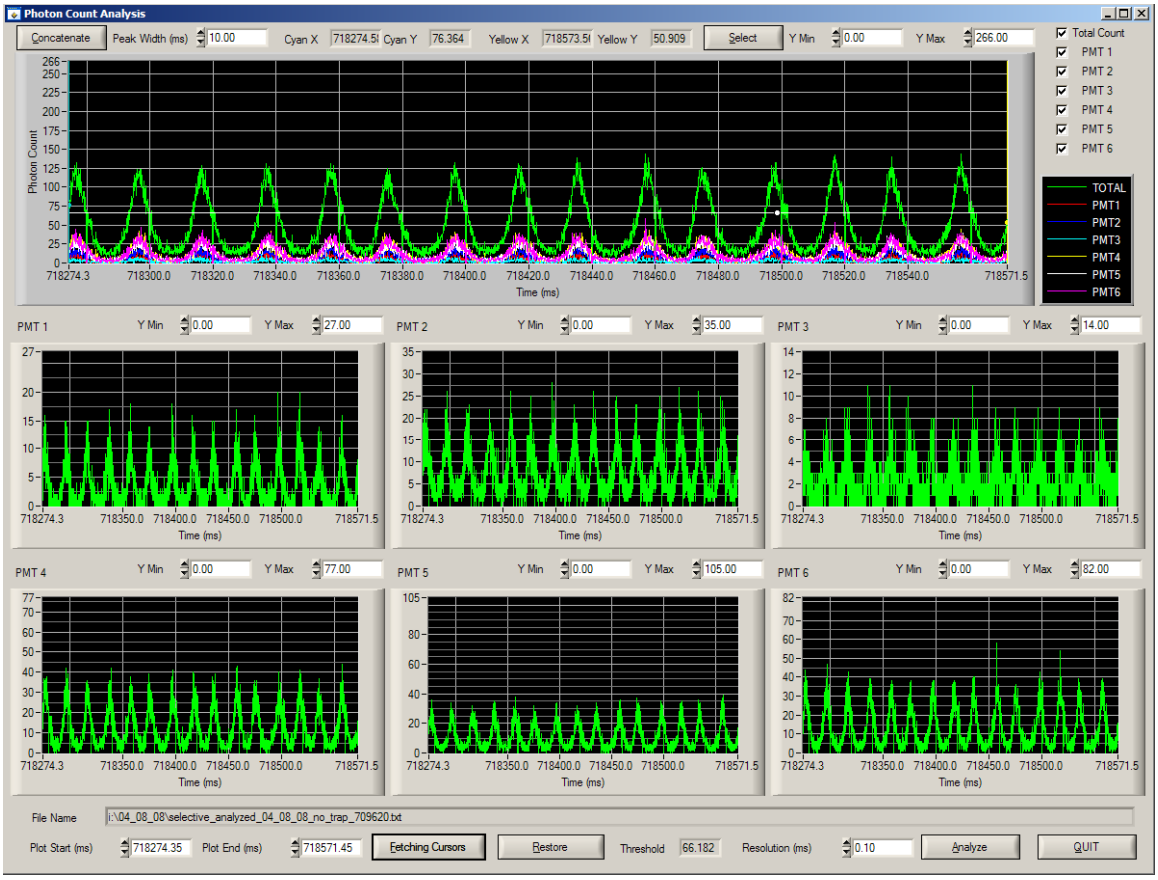


Figure 5.22 : Photon count analysis in ASMOS

Normally we have a photon count peak whenever a droplet is trapped within the excitation UV laser beam. Since droplets are generated and trapped periodically, we expect synchronized periodic peaks in an optimal working condition. The concatenation cuts off the unnecessary photon parts (background scattering, auto-fluorescence, noises, etc.) which are away from peaks and below a threshold. The concatenation can start by clicking "Concatenate" button on the photon count analysis panel. One can set the value of a peak width as a user input on the panel.

Chapter 6 Computer simulation of the whole trajectory of a droplet in the lens cube assembly

6.1 Introduction

To guarantee the successful operation of ASMOS and a high throughput single molecule detector, I have simulated the whole trajectory of a droplet in the lens cube assembly. The simulation result is used to determine the optimal operational conditions for the instruments. The trajectory of a droplet is affected by IR laser guidance, water evaporation, Brownian motion, etc.

In this chapter, I explain the details of these effects and integration into a simulation program written in C language. I present how a droplet is guided into the excitation UV focused region by the radiation force exerted by the IR guiding lasers, and how fast the size of the droplet is reduced by the evaporation effect.

6.2 Calculation of Infrared laser guidance

6.2.1 Generalized Lorenz-Mie Theory (GLMT)

"All problems in theoretical optics are problems in Maxwell's theory and should be treated as such when a full, formal solution is required. … The scattering of light a homogeneous sphere cannot be treated in a general way, other than by the formal solution of Maxwell's equations with the appropriate boundary conditions." [123]

As mentioned by van de Hulst above, the rigorous calculation of the optical scattering begins with Maxwell's equations. In 1908-1909, Mie and Debye published pioneering papers on the scattering of a plane wave by a spherical particle,[124, 125] inspired by even earlier work by Lorenz in 1890.[126] Thanks to their contributions, the

classical Lorenz-Mie Theory (LMT) was established for the scattering of a plane wave by a spherical particle, and is still valid for arbitrary particle size, refractive index, and wavelength.[118]

However, LMT is based on a plane wave which is just an approximation in reality. For example, the popular TEM mode laser produces a Gaussian beam and can not be treated as a plane wave unless it is emanating from a far distance.[118, 127] The Generalized Lorenz-Mie Theory (GLMT) was developed by Gouesbet *et al.* to extend LMT for arbitrarily shaped beams.[128-131] A beam shape coefficient, g_n^m is introduced and contains information about the incident beam profile.[132] An infinite set of beam shape coefficients is at the core of GLMT devoted to the scattering of an arbitrary shaped beam by spheres.[133, 134] A localized approximation is introduced for the fast calculation of beam shape coefficients, g_n^m [134, 135], and has been justified rigorously for the case of Gaussian beams [136] and for arbitrarily shaped beams.[131] GLMT predictions have been previously compared to experimental results and were in good agreement within a few percent, in all size regimes.[118, 137-140]

6.2.2 Radiation force by the IR guiding lasers expressed in GLMT

Every incident photon carries momentum. The radiation force exerted by the IR guiding lasers on a droplet is proportional to the net momentum removed from the incident photons by the scattering.[129] The radiation force vector in GLMT is expressed as

$$\vec{F}(\vec{r}) = \left(\frac{n_m}{c}\right)\frac{2P}{\pi\omega_0{}^2}\left[\hat{x}C_{pr,x}(\vec{r}) + \hat{y}C_{pr,y}(\vec{r}) + \hat{z}C_{pr,z}(\vec{r})\right] \tag{6.1}$$

where $C_{pr,x}, C_{pr,y}, C_{pr,z}$ are the cross sections for radiation pressure in the Cartesian coordinate system and defined in reference [129, 130]. And c is the speed of light, n_m is a

refractive index of the air, ω_0 is a beam radius at the focal point (the beam waist), and P is an incident laser beam power. The radiation force components are characterized as two different categories, the longitudinal radiation force along z axis (the main axial direction of propagation), and the transverse radiation force along x and y axes (the radial directions).[138] When the displacement of the droplet from the focus is less than its radius, the radiation force is approximately proportional to the droplet displacement in either case.

6.2.3 Longitudinal radiation force (z direction)

Gouesbet *et al.* rigorously formulated the radiation forces in GLMT. It consists of massive sets of complex equations and can be solved by numerical methods only. Here I briefly present the core formulas of GLMT for the Gaussian beam. The cross section for radiation pressure in z direction is given by the references [129, 130] as,

$$C_{pr,z} = \frac{\lambda^2}{\pi}\sum_{n=1}^{\infty}\sum_{p=-n}^{+n}\left\{\begin{array}{l}\dfrac{1}{(n+1)^2}\dfrac{(n+1+|p|)!}{(n-|p|)!}\mathrm{Re}\left[\begin{array}{l}(a_n + a_{n+1}^{*} - 2a_n a_{n+1}^{*})g_{n,TM}^{p} g_{n+1,TM}^{p*} \\ +(b_n + b_{n+1}^{*} - 2b_n b_{n+1}^{*})g_{n,TE}^{p} g_{n+1,TE}^{p*}\end{array}\right] \\ \\ +p\dfrac{2n+1}{n^2(n+1)^2}\dfrac{(n+|p|)!}{(n-|p|)!}\mathrm{Re}\left[i(2a_n b_n^{*} - a_n - b_n^{*})g_{n,TM}^{p} g_{n,TE}^{p*}\right]\end{array}\right\} \tag{6.2}$$

where λ is the wavelength of an incident beam in the air. The asterisk (*) indicates the complex conjugate. The scattering coefficients of the LMT, a_n and b_n are defined as,

$$a_n = \frac{\psi_n(\alpha)\psi_n'(\beta) - M\psi_n'(\alpha)\psi_n(\beta)}{\xi_n(\alpha)\psi_n'(\beta) - M\xi_n'(\alpha)\psi_n(\beta)} \tag{6.3}$$

$$b_n = \frac{M\psi_n(\alpha)\psi_n'(\beta) - \psi_n'(\alpha)\psi_n(\beta)}{M\xi_n(\alpha)\psi_n'(\beta) - \xi_n'(\alpha)\psi_n(\beta)} \tag{6.4}$$

where the prime indicates the derivative of the function for the argument in the parentheses, and the size parameter α is,

$$\alpha = \frac{\pi d}{\lambda} = kr \tag{6.5}$$

Also

$$\beta = M\alpha \tag{6.6}$$

$$M = \frac{k_{droplet}}{k} = \sqrt{\frac{\varepsilon_{droplet}}{\varepsilon_{air}}} \tag{6.7}$$

and

$$k = \frac{2\pi}{\lambda} \tag{6.8}$$

Here, k is the angular wave number of the incident beam in the air, d is the diameter, r is the radius of the droplet, M is the complex refractive index of a droplet relative to the air, and ε is the electric permittivity of the medium, respectively. We assume a non-magnetic droplet, which means the magnetic permeability is,

$$\frac{\mu_{droplet}}{\mu_{air}} = 1 \tag{6.9}$$

On the other hand, $\psi_n^1(kr)$ is the spherical Bessel function given by,

$$\psi_n^1(kr) = \sqrt{\frac{\pi}{2kr}} J_{n+\frac{1}{2}}(kr) \tag{6.10}$$

where $J_{n+\frac{1}{2}}(kr)$ is a Bessel function of half-integer order. Gouesbet *et al.* introduced the Ricatti-Bessel functions, $\psi_n(kr)$ and $\xi_n(kr)$ as, [129, 130]

$$\begin{aligned}\psi_n(kr) &= kr\psi_n^1(kr) \\ &= \sqrt{\frac{\pi kr}{2}} J_{n+\frac{1}{2}}(kr)\end{aligned}$$

$$\begin{aligned}\xi_n(kr) &= \psi_n(kr) + i(-1)^n \sqrt{\frac{\pi kr}{2}} J_{-n-\frac{1}{2}}(kr) \\ &= \sqrt{\frac{\pi kr}{2}} H^{(2)}_{n+\frac{1}{2}}(kr)\end{aligned} \tag{6.11}$$

where $H^{(2)}_{n+\frac{1}{2}}(kr)$ is the Hankel function and valid for the case of a half-integer order only.

Now g_n^m, the beam shape coefficients are determined as the followings.

$$g^m_{n,TM} = \frac{1}{C_n^{pw}} \frac{(2n+1)^2}{\pi n(n+1)} \frac{(n-|m|)!}{(n+|m|)!} \int_0^{\pi}\int_0^{\infty} \left[\begin{array}{l} \left\{ \dfrac{F}{2} \left(\displaystyle\sum_{j_+=m}^{jp} \psi_{jp} + \sum_{j_-=m}^{jp} \psi_{jp} \right) + x_0 G \sum_{j_0=m}^{jp} \psi_{jp} \right\} \\ \times \quad r\psi_n^1(kr) P_n^{|m|}(\cos\theta) \sin\theta d\theta d(kr) \end{array} \right] \tag{6.12}$$

$$g^m_{n,TE} = \frac{1}{C_n^{pw}} \frac{(2n+1)^2}{\pi n(n+1)} \frac{(n-|m|)!}{(n+|m|)!} \int_0^{\pi}\int_0^{\infty} \left[\begin{array}{l} \left\{ \dfrac{F}{2} \left(-i \displaystyle\sum_{j_+=m}^{jp} \psi_{jp} + i \sum_{j_-=m}^{jp} \psi_{jp} \right) + y_0 G \sum_{j_0=m}^{jp} \psi_{jp} \right\} \\ \times \quad r\psi_n^1(kr) P_n^{|m|}(\cos\theta) \sin\theta d\theta d(kr) \end{array} \right]$$

where

$$C_n^{pw} = \frac{1}{k} i^{n-1} (-1)^n \frac{2n+1}{n(n+1)} \tag{6.13}$$

$$\psi_{jp} = \left(\frac{iQr\sin\theta}{\omega_0^2} \right)^j \frac{(x_0 - iy_0)^{j-p} (x_0 + iy_0)^p}{(j-p)!\, p!} \tag{6.14}$$

and

$$F = \psi_0^0 \sin\theta \left(1 - \frac{2Q}{l} r\cos\theta \right) \exp(K) \tag{6.15}$$

$$G = \psi_0^0 \frac{2Q}{l} \cos\theta \exp(K) \tag{6.16}$$

$$\begin{aligned} j_+ &= j + 1 - 2p = j_0 + 1 \\ j_- &= j - 1 - 2p = j_0 - 1 \end{aligned} \tag{6.17}$$

The symbol $\sum_{c}^{jp}$ designates the sum $\sum^{jp} = \sum_{j=0}^{\infty} \sum_{p=0}^{j}$ restricted to the condition c.[129]

Finally again,

$$K = -ik(r\cos\theta - z_0) \tag{6.18}$$

$$\psi_0^0 = iQ \exp\left(-iQ \frac{r^2 \sin^2\theta}{\omega_0^2} \right) \exp\left(-iQ \frac{x_0^2 + y_0^2}{\omega_0^2} \right) \tag{6.19}$$

$$Q = \frac{1}{i + 2(\zeta - \zeta_0)} \tag{6.20}$$

where $\zeta = \frac{z}{l}$, $\zeta_0 = \frac{z_0}{l}$, and l is the so-called diffraction or spreading length, $l = k\omega_0^2$. (x_0, y_0, z_0) is the coordinate of the beam waist center and (x, y, z) is the coordinate of the incident beam, while (r, θ, φ) is the coordinate of the scattered light, and all of these are viewed from the particle center.

6.2.4 Transverse radiation force (x and y directions)

The cross sections for radiation pressure in x and y directions are given by the references [129, 130] as,

$$C_{pr,x} = \mathrm{Re}(C_{pr,xy}) \tag{6.21}$$

$$C_{pr,y} = \mathrm{Im}(C_{pr,xy}) \tag{6.22}$$

where

$$C_{pr,xy} = \frac{\lambda^2}{2\pi}\sum_{p=1}^{\infty}\sum_{n=p}^{\infty}\sum_{m=p-1\neq 0}^{\infty}\frac{(n+p)!}{(n-p)!}\left[\begin{array}{c}\left(S_{mn}^{p-1}+S_{nm}^{-p}-2U_{mn}^{p-1}-2U_{nm}^{-p}\right)\left(\frac{1}{m^2}\delta_{m,n+1}-\frac{1}{n^2}\delta_{n,m+1}\right)+\\ \frac{2n+1}{n^2(n+1)^2}\delta_{nm}\left(T_{mn}^{p-1}-T_{nm}^{-p}-2V_{mn}^{p-1}+2V_{nm}^{-p}\right)\end{array}\right] \quad (6.23)$$

and

$$U_{nm}^{p} = a_n a_m^{*} g_{n,TM}^{p} g_{m,TM}^{p+1\ *} + b_n b_m^{*} g_{n,TE}^{p} g_{m,TE}^{p+1\ *} \quad (6.24)$$

$$V_{nm}^{p} = ib_n a_m^{*} g_{n,TE}^{p} g_{m,TM}^{p+1\ *} - ia_n b_m^{*} g_{n,TM}^{p} g_{m,TE}^{p+1\ *} \quad (6.25)$$

$$S_{nm}^{p} = (a_n + a_m^{*}) g_{n,TM}^{p} g_{m,TM}^{p+1\ *} + (b_n + b_m^{*}) g_{n,TE}^{p} g_{m,TE}^{p+1\ *} \quad (6.26)$$

$$T_{nm}^{p} = -i(a_n + b_m^{*}) g_{n,TM}^{p} g_{m,TE}^{p+1\ *} + i(b_n + a_m^{*}) g_{n,TE}^{p} g_{m,TM}^{p+1\ *} \quad (6.27)$$

The scattering coefficients of the LMT, a_n and b_n are defined in equation (6.3) and equation (6.4). g_n^m, the beam shape coefficients are given in equation (6.12).

6.2.5 A localized approximation for the fast calculation

The original equation sets in GLMT include infinite sums and integrations of many functions, which require huge amount of calculation time. Gouesbet *et al.* developed a localized approximation for the fast calculation of g_n^m, the beam shape coefficients.[134] I'll present their results in this section shortly.

$$\begin{pmatrix} g_{n,TM}^{m} \\ g_{n,TE}^{m} \end{pmatrix} = \frac{1}{2(1+2iz_0^{+})}\exp\left(\frac{iz_0^{+}}{s^2} - \frac{x_0^{+2}+y_0^{+2}}{1+2iz_0^{+}}\right)\exp\left[-\frac{(n+\frac{1}{2})^2}{1+2iz_0^{+}}s^2\right] R_n^m (-i)^{|m|}\begin{pmatrix} iF_{n,TM}^{m} \\ F_{n,TE}^{m} \end{pmatrix} \quad (6.28)$$

where the dimensionless coordinates are defined as,

$$x_0^+ = \frac{x_0}{\omega_0}$$
$$y_0^+ = \frac{y_0}{\omega_0} \tag{6.29}$$
$$z_0^+ = \frac{z_0}{l}$$

Respectively, ω_0 is the beam waist radius, $l = k\omega_0^2$ is the spreading length, and $s = \frac{\omega_0}{l}$ is a dimensionless parameter. (x_0, y_0, z_0) is the coordinate of the beam waist center, viewed from the particle center. Also

$$R_n^m = \left(\frac{2}{2n+1}\right)^{|m|-1}, \quad m \neq 0$$
$$R_n^0 = \frac{2n(n+1)}{2n+1}, \quad m = 0 \tag{6.30}$$

and

$$\begin{pmatrix} F_{n,TM}^0 \\ F_{n,TE}^0 \end{pmatrix} = \begin{pmatrix} 2x_0^+ \\ 2iy_0^+ \end{pmatrix} \sum_{j=0}^{\infty} a^{2j+1} \frac{X_-^j \cdot X_+^j}{j!(j+1)!} \quad (m=0)$$

$$\begin{pmatrix} F_{n,TM}^m \\ F_{n,TE}^m \end{pmatrix} = a^{m-1} \frac{X_-^{m-1}}{(m-1)!} + \sum_{j=m}^{\infty} a^{2j-m+1} \frac{X_+^{j-m} \cdot X_-^j}{(j-m)!\, j!} \begin{pmatrix} \frac{X_+}{j-m+1} + \frac{X_-}{j+1} \\ \frac{X_+}{j-m+1} - \frac{X_-}{j+1} \end{pmatrix} \quad (m>0) \tag{6.31}$$

$$\begin{pmatrix} F_{n,TM}^{-|m|} \\ -F_{n,TE}^{-|m|} \end{pmatrix} = a^{|m|-1} \frac{X_+^{|m|-1}}{(|m|-1)!} + \sum_{j=|m|}^{\infty} a^{2j-|m|+1} \frac{X_+^j \cdot X_-^{j-|m|}}{j!(j-|m|)!} \begin{pmatrix} \frac{X_-}{j-|m|+1} + \frac{X_+}{j+1} \\ \frac{X_-}{j-|m|+1} - \frac{X+}{j+1} \end{pmatrix} \quad (m<0)$$

By definition,

$$X_- = x_0^+ - iy_0^+$$
$$X_+ = x_0^+ + iy_0^+ \tag{6.32}$$

Here g_n^m, the beam shape coefficients also have infinite sums, but it converges

much faster than the original formulas, so significantly decreases computation time (>100 times faster).

6.3 Evaporation of water from a droplet

6.3.1 Introduction

In 1959, Fuchs mentioned about the evaporation process for droplets in the preface of his survey.[141] It describes how difficult the complete description of evaporation process for droplets is.

"Under natural conditions this phenomenon is extremely complex. The bulk of the droplet evaporates almost immediately. The process is non-stationary and occurs in a medium with unequal temperature and vapor concentration. The drops move irregularly relative to the medium and are more or less deformed, while circulation arises within the drops. Heat transfer between the drops and the medium occurs by three different mechanisms (conduction, convention and radiation)."[142]

In 1949, Kinzer *et al.* described evaporation from spherical droplet in terms of heat and vapor transferred and calculated the temperature of a freely falling water droplet.[143] In 1971, Duguid *et al.* determined the evaporation rates of small, freely falling water droplets by recording the drop at fixed time intervals, and compared their results with Kinzer and Gunn's and the original mass diffusion theory by Maxwell.[144] Surprisingly, Duguid *et al.* showed that the evaporation of pure water droplets is best described by simple mass diffusion theory by Maxwell, with a small ventilation effect.[144] However, all of these approaches do not perfectly catch up the experimental results.

For a computer simulation of the evaporation from a droplet, I assume a pure water droplet for simplicity. It is a reasonable assumption since the main composition of a droplet is still pure water although it contains a single protein molecule and some buffer

molecules. Furthermore, I combine Maxwell's mass diffusion theory with Kinzer's approach with the heat transfer, in steady state equilibrium. By combining both of them together, we get more realistic results for the evaporation from a droplet.

6.3.2 Mass diffusion

Diffusion theory was proposed by Maxwell in 1877. It describes evaporation of water from a droplet as a pure diffusion process of the water molecules through the surrounding medium (air). It assumes the evaporation is a steady-state equilibrium process, and is given by

$$\frac{dm}{dt} = 4\pi aD(\rho_\infty - \rho_a) \tag{6.33}$$

where m is mass of a droplet, a is radius of a droplet, D is a diffusion coefficient of water vapor in the surrounding gaseous medium (air), ρ_∞ is a density of water vapor in the air, and ρ_a is a density of water vapor at the surface of a droplet.[144]

Since $m = \frac{4}{3}\pi\rho_l a^3$ for a homogenous water droplet where ρ_l is a density of liquid water in a droplet, the equation (6.33) can be rephrased as

$$\frac{4}{3}\pi\rho_l\left(3a^2\frac{da}{dt}\right) = 4\pi aD(\rho_\infty - \rho_a) \tag{6.34}$$

Therefore the evaporation rate may be written as in terms of radius,

$$2a\frac{da}{dt} = \frac{da^2}{dt} = \frac{2D}{\rho_l}(\rho_\infty - \rho_a) \tag{6.35}$$

6.3.3 Heat transfer

The transport of water vapor by diffusion is a molecular process closely related to the diffusion of heat.[143] Whenever water evaporates from a droplet as a diffusion process, it takes the latent heat away from the droplet lowering the surface temperature.

$$dQ = L\, dm \tag{6.36}$$

where Q is the amount of energy required to change the phase of water from liquid to gas, and L is the latent heat of evaporation for water. The heat energy taken away from the droplet is used to change the phase of water from liquid to gas.

Therefore a heat loss, Q_{loss} and the corresponding heat loss flux, $\frac{1}{4\pi a^2}\frac{dQ_{loss}}{dt}$ from a droplet is expressed as, with the help of equation (6.33)

$$\frac{1}{4\pi a^2}\frac{dQ_{loss}}{dt} = \frac{1}{4\pi a^2} L \frac{dm}{dt} = \frac{DL}{a}(\rho_\infty - \rho_a) \tag{6.37}$$

At the same moment, since a droplet is getting cooled down during the evaporation, the temperature gradient at the surface of the droplet causes a heat gain, Q_{gain} into the droplet. In addition I included an additional heat gain source, absorption of the IR guiding laser beams by the droplet. (See [Chapter 6.3.7] for the details) By applying the heat diffusion equation we can set up a differential equation for the heat gain flux,

$$\frac{1}{4\pi a^2}\frac{dQ_{gain}}{dt} = \frac{1}{4\pi a^2}\left\{4\pi a K (T_\infty - T_a) + \alpha\right\} \tag{6.38}$$

where K is the coefficient of thermal conductivity, T_∞ is the temperature of the

surrounding air, T_a is the temperature of the surface of a droplet, and α is the heat energy absorbed from the IR guiding laser beams by the droplet.

6.3.4 In the steady state

In the steady state, there is no net heat flux between a droplet and the surround air. Therefore the net heat flux, a sum of the heat loss flux [Equation (6.37)] and the heat gain flux [Equation (6.38)] must be zero.

$$\frac{1}{4\pi a^2}\left(\frac{dQ_{loss}}{dt}+\frac{dQ_{gain}}{dt}\right)=0 \tag{6.39}$$

In a more explicit form,

$$\frac{DL}{a}(\rho_\infty-\rho_a)+\frac{1}{4\pi a^2}\{4\pi aK(T_\infty-T_a)+\alpha\}=0 \tag{6.40}$$

By defining a new constant Γ as $\Gamma\equiv\frac{K}{DL}$, we reach

$$T_\infty-T_a=\frac{1}{\Gamma}(\rho_a-\rho_\infty)-\frac{\alpha}{4\pi aK} \tag{6.41}$$

6.3.5 Evaporation rate

The evaporation rate of water from a droplet is already given by a diffusion theory by Maxwell in the equation (6.35).

$$\frac{da^2}{dt}=\frac{2D}{\rho_l}(\rho_\infty-\rho_a)$$

Since $\rho_\infty - \rho_a = \Gamma\left(T_a - T_\infty - \frac{\alpha}{4\pi aK}\right)$ from rephrasing the equation (6.41) in the steady state, the evaporation rate of water is presented as, in terms of radius of a droplet

$$\frac{da^2}{dt} = -\frac{2D\Gamma}{\rho_l}\left(T_\infty - T_a + \frac{\alpha}{4\pi aK}\right) \tag{6.42}$$

The equation (6.42) is the most important equation for simulating evaporation of water from a droplet. What we need to do is solve this differential equation numerically for a, the radius of a droplet [Chapter 6.5.2], by using a numerical root finding for T_a, the temperature at the surface of the droplet [Equation (6.47) in Chapter 6.3.6], and calculating α, absorbed energy by water from IR laser guiding beams [Equation (6.54) in Chapter 6.3.7]. Other parameters are assumed to be constants for simplicity.

6.3.6 Numerical root finding for T_a, the temperature at the surface of a droplet during evaporation

Providing the ideal gas law of $PV = nRT$, the density of water vapor becomes

$$\rho = \frac{nM}{V} = \frac{M}{R}\left(\frac{P}{T}\right) \tag{6.43}$$

where M is molecular weight of water. By combining the equation (6.41) with the equation (6.43), we get

$$T_\infty - T_a = \frac{M}{\Gamma R}\left(\frac{P_a}{T_a} - \frac{P_\infty}{T_\infty}\right) - \frac{\alpha}{4\pi aK} \tag{6.44}$$

where P_a is a pressure of water vapor at the surface of a droplet, and P_∞ is a pressure of water vapor in the surrounding air.

Next, we rearrange equation (6.44) to define a new constant X for a quadratic expression in terms of T_a.

$$T_a + \left(\frac{M}{\Gamma R}\right)\frac{P_a}{T_a} = T_\infty + \left(\frac{M}{\Gamma R}\right)\frac{P_\infty}{T_\infty} + \frac{\alpha}{4\pi a K} \equiv X \tag{6.45}$$

Note that the right side of equation (6.45) doesn't have any dependence on T_a. However, it is a function of a, the radius of the droplet for example. So the simulation updates the value of X whenever the radius changes. In addition, we define the relative humidity of air (RH) as

$$\mathrm{RH} \equiv \frac{P_\infty}{P_{dew,\infty}} \tag{6.46}$$

where $P_{dew,\infty}$ is a pressure of water vapor in the surrounding air, at the dew point of T_∞.

Finally, we multiply T_a into the both sides of the equation (6.45) to make a quadratic equation in terms of T_a.

$$T_a^2 - XT_a + \left(\frac{M}{\Gamma R}\right)P_a = 0 \tag{6.47}$$

where $X = T_\infty + \left(\frac{M}{\Gamma R}\right)\frac{P_\infty}{T_\infty} + \frac{\alpha}{4\pi a K}$. We need to calculate T_a by using a numerical root finding algorithm,[145] because P_a, the pressure of water vapor at the surface of a droplet, is strongly correlated with T_a, the temperature at the surface of the droplet. The

correlation between P_a and T_a has been published in CRC handbook in detail.[146]

6.3.7 An absorbed heat energy by a droplet from IR guiding laser illumination

In this calculation, I assume a simple plane wave for the incident beam. Although the IR guiding laser beams have a Gaussian beam shape, this assumption gives a good approximation since we're treating the small size of a droplet only. Figure 6.1 shows the trajectory of the IR guiding laser beam inside a droplet.

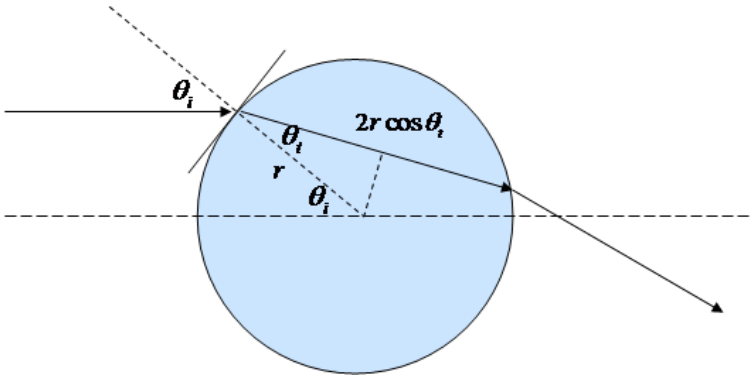


Figure 6.1 : Trajectory of the IR guiding laser beam inside a droplet

The refraction of the laser beam follows Snell's Law, $n_i \sin\theta_i = n_t \sin\theta_t$ and the beam propagates a distance of $l \equiv 2r\cos\theta_t$ through the inside of a droplet.

First of all, I begin with the total energy of incident beams, which is obtained as

$$P_{incident} = I_0 \int_0^r (2\pi y)dy = I_0 \pi r^2 \tag{6.48}$$

where I_0 is the intensity of the incident beam through the center of a droplet, r is the radius of a droplet, and $y = r\sin\theta_i$.

The transmittance of the incident beam with an incident angle of θ_i is defined as

$$T \equiv \frac{1}{2}(T_{\parallel} + T_{\perp}) \tag{6.49}$$

where

$$T_{\parallel} = \frac{\sin 2\theta_i \sin 2\theta_t}{\sin^2(\theta_i + \theta_t)\cos^2(\theta_i - \theta_t)} \tag{6.50}$$

$$T_{\perp} = \frac{\sin 2\theta_i \sin 2\theta_t}{\sin^2(\theta_i + \theta_t)} \tag{6.51}$$

and θ_t is an angle of the refracted beam.[127] The refraction of the incident beam follows Snell's law, $n_i \sin\theta_i = n_t \sin\theta_t$. Therefore the total energy of refracted beams just after the first interface is,

$$P_{refracted} = I\int_0^r T(2\pi y)dy \tag{6.52}$$

Next, the refracted beam propagates a distance of $l \equiv 2r\cos\theta_t$ through the inside a droplet. Due to the IR absorption by water, the intensity of the refracted beam decreases with a rate of

$$e^{-\alpha l} = e^{-\alpha 2r\cos\theta_t} \tag{6.53}$$

where α is the absorption coefficient of water at 830 nm, as shown in [Figure 6.2].

Therefore the total absorbed energy by water from the refracted beams can be formulated as,

$$\begin{aligned} P_{absorbed} &= P_{refracted} \quad - \quad I_0\int_0^r Te^{-\alpha l}(2\pi y)dy \\ &= I_0\int_0^{\frac{\pi}{2}} T(1-e^{-\alpha 2r\cos\theta_t})(2\pi r\sin\theta_i)(r\cos\theta_t d\theta_i) \end{aligned} \tag{6.54}$$

where $n_i \sin\theta_i = n_t \sin\theta_t$. Multiple reflections inside the droplet are neglected for

simplicity.

In addition, the ratio of absorbed energy to the incident energy is,

$$\frac{P_{absorbed}}{P_{incident}} = \int_0^{\frac{\pi}{2}} T(1 - e^{-\alpha 2r\cos\theta_t}) \sin 2\theta_i d\theta_i \tag{6.55}$$

In the simulation, the Romberg integration has been performed for the fast and reliable integration.[145]

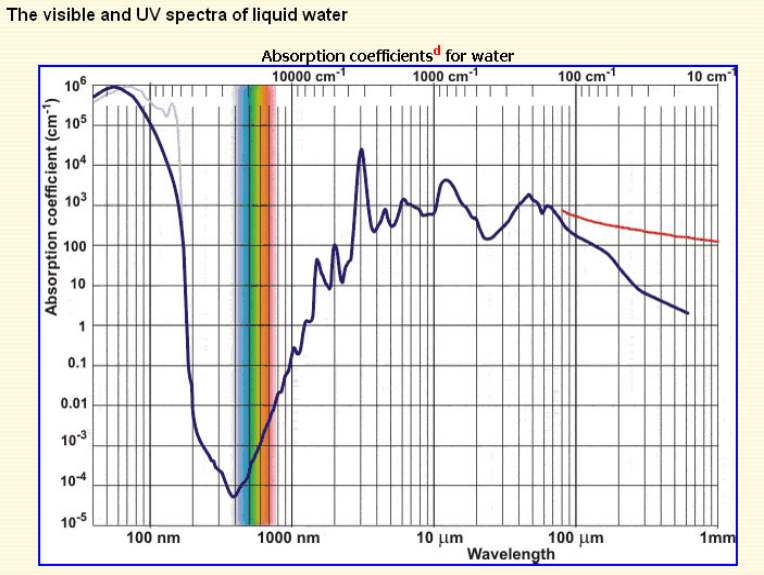


Figure 6.2 : Absorption coefficients for liquid water

From reference [147]. The data are extracted from references [148-151].

6.4 Brownian motion

Small particles in a fluid perpetually move about in a random manner. It was first observed by Brown, and was explained theoretically by Einstein in 1905 [152] from the random collisions of the particle with the molecules of the liquid.[153] In this section, I present how to simulate the Brownian motion of a droplet in the air. I introduce an efficient method to simulate the Brownian motion, by modeling it as a vector white noise process according to the references of [154-157].

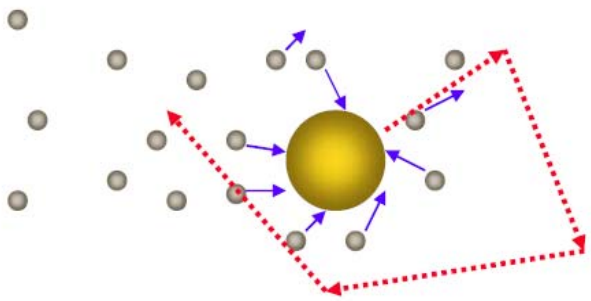


Figure 6.3 : Brownian motion process

From ref. [154]: G. Ahmadi, lecture note for ME437/537, Clarkson University

First of all, we begin with the Knudson number K_n, the ratio of the gas mean free path to particle radius.

$$K_n = \frac{2\lambda}{d} \tag{6.56}$$

where d is the droplet diameter and λ is the molecular mean free path of the air molecule. The mean free path of the air was estimated as 100 nm since the typical molecular diameter of air molecules is 0.3 nm. [153] Then the corresponding Stokes-Cunningham slip correction C_c is given by Abuzeid *et al*. [155]

$$C_c = 1 + 1.257K_n + 0.40K_n e^{-(1.1/K_n)} \tag{6.57}$$

Next, the Brownian motion is governed by the Langevin equation,

$$\frac{d\dot{x}}{dt} + \beta\dot{x} = n_x(t) \tag{6.58}$$

where $\dot{x}$ is the velocity of the droplet in x direction and n(t) is the effective Brownian force. The coefficient β for the damping is defined as,

$$\beta = \frac{3\pi\mu d}{C_c m} \tag{6.59}$$

where μ is the viscosity of the air and m is the mass of the droplet.

Calculation of the effective Brownian force, n(t) is based on the pioneering studies of a Gaussian white noise random process.[158-160] The spectral intensity of the noise, S_{nn} is given by

$$S_{nn} = \frac{2k_B T \beta}{\pi m} \tag{6.60}$$

where k_B is the Boltzmann constant. By generating independent Gaussian random numbers Z_i having unit variance and zero mean, the amplitude of the Brownian force in the x direction is calculated as

$$n_x(t_i) = Z_i \sqrt{\frac{\pi S_{nn}}{\Delta t}} \tag{6.61}$$

where t_i is the current time in the simulation and Δt is the time step of the simulation. Δt should be much larger than the molecular time scale and much smaller than the particle relaxation time.[156] I repeat the same procedures for the y and z directions in the simulation. Figure 6.4 presents the numerically simulated Brownian force.

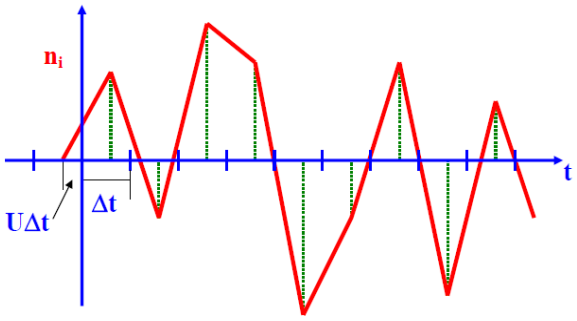


Figure 6.4 : Numerically simulated Brownian force

From ref. [154]: G. Ahmadi, lecture note for ME437/537, Clarkson University

The following C code executes the calculation of the amplitude of the Brownian force in one direction.

```
double brownian_amplitude(double dt, double radius) {
        double mean_free_path, Kn, Cc, beta, Snn;
        double number_density_mixture, number_density_water, number_density_ethylene_glycol;
        double mole_fraction;
        double diffusion_coefficient;
        double brownian_amplitude;

        number_density_water = DENSITY_H2O / MOLECULAR_WEIGHT_H2O *  AvogadroConstant();
        number_density_ethylene_glycol = DENSITY_ETHYLENE_GLYCOL /
                MOLECULAR_WEIGHT_ETHGLY * AvogadroConstant();
        number_density_mixture = g_EthyleneGlycol * number_density_ethylene_glycol + (1.0-g_EthyleneGlycol)
                * number_density_water;

        // http://hyperphysics.phy-astr.gsu.edu/hbase/kinetic/menfre.html#c3
        mean_free_path = 1.0 / (number_density_mixture * sqrt(2.0) * ( Pi() * DIAMETER(radius) *
                DIAMETER(radius)) );

        // Abuzeid et al, Wall deposition of aerosol particles in a turbulent channel flow.pdf
        Kn = 2.0 * mean_free_path / DIAMETER(radius);           // Knudson Number
        Cc = 1.0 + 1.257*Kn + 0.40*Kn*exp(-1.1/Kn);             // Cunningham Slip Correction (Cc ~ 1.0)
        g_Cc = Cc;

        /*      Dumont et al, Solvent-tuning the collapse and helix formation time scales of lambda 6-85 (2006)
        x = 0.2078963606   (mole fraction of ethylene glycol)       T : in Celcius
        (0.004757 + 0.047x) {1 + (221 + 573x)Exp[-0.048T] + (154 - 69x)Exp[-0.01T]} */

        mole_fraction = (g_EthyleneGlycol*number_density_ethylene_glycol) / number_density_mixture;
        g_viscosity = (1.0e-3)*(0.004757 + 0.047*mole_fraction)*(1.0 + (221.0 + 573.0*mole_fraction) *
                        exp(-0.048*KelvinToCelsius(g_Temperature)) + (154.0 - 69.0*mole_fraction) *
                        exp(-0.01*KelvinToCelsius(g_Temperature)));
        diffusion_coefficient = Cc * (BOLTZMANN_COEFF*g_Temperature) / (6.0*Pi()*g_viscosity*radius);

        // G. Ahmadi, ME437/537 Lecture Note, Clarkson University
        beta = (3.0 * Pi() * g_viscosity * DIAMETER(radius)) / (Cc * MASS);
        Snn = (2.0 * BOLTZMANN_COEFF * g_Temperature) * beta / (Pi() * MASS);
        brownian_amplitude = sqrt( Pi() * Snn / dt );

        return brownian_amplitude;
}
```

6.5 Simulation results

6.5.1 Radiation force by guiding lasers predicted by GLMT

To verify the validity of the laser guidance, I performed a computer simulation of infrared laser guidance by using Generalized Lorenz-Mie Theory (GLMT) [118, 129,

161] and considering Brownian Motions of a droplet in the air [155-157]. I neglected the evaporation of water for simplicity, to mainly understand how guiding lasers work near the beam center. The simulation program was written in C language, under the National Instruments LabWindows/CVI 8.5 GUI environment.

In a small displacement of less than 10 μm, two co-aligned guiding laser beams generate a gradient restoring force, which is proportional to the deviation from the center of beam as shown in [Figure 6.5]. Thus the laser guidance predicted by GLMT simply acts like a Hookean spring near the center of beam. The laser guiding simulation shows that the micron sized drops can be reliably well trapped by the guiding lasers. The initial diameter of a droplet was 10 μm, with two IR guiding lasers having 100 mW beam power and 10 μm beam waist.

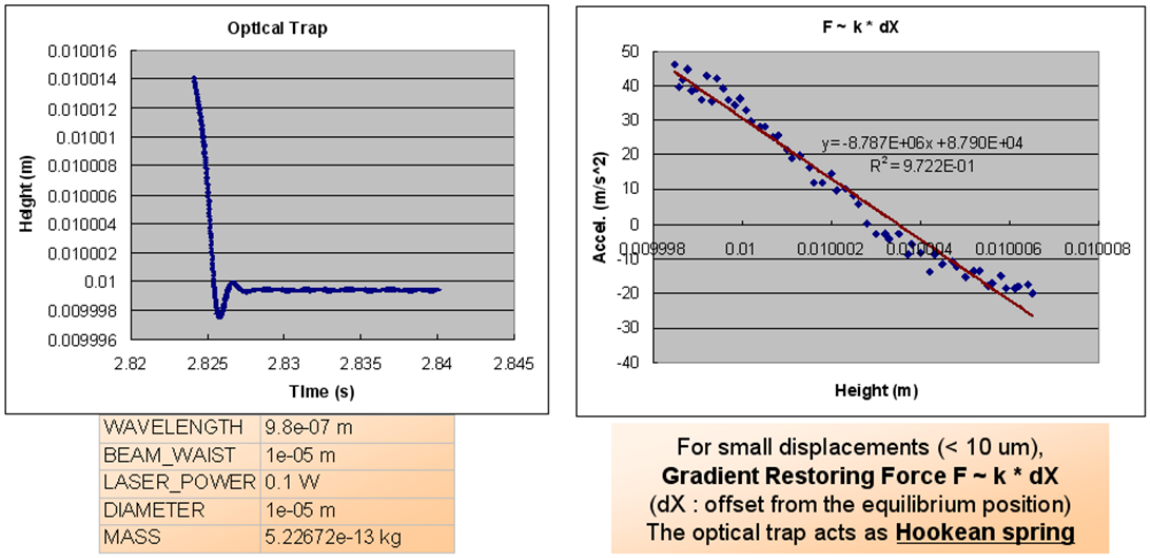


Figure 6.5 : Simulation of the laser guidance (left) and restoring force by guiding lasers (right)

The left plot shows the simulated (height) trajectory of a droplet by the laser guidance. The droplet is well trapped around the center of beam (at the height of 1 cm) with a tiny deviation of less than 1 μm.

6.5.2 The evaporation of water from a droplet

For the next step, the evaporation of water from a droplet has been simulated according to the steady-state equation (6.42) in [Chapter 6.3.5]. I applied two Gaussian

IR beams of 830 nm wavelength, with a beam power of 100 mW and a beam waist of 70 μm, for the IR laser illumination. The evaporation rate mainly depends on the relative humidity, since IR absorption of water at 830 nm is quite small. Thus, the radius change of a droplet is mainly affected by the relative humidity, as shown in [Table 6.1].

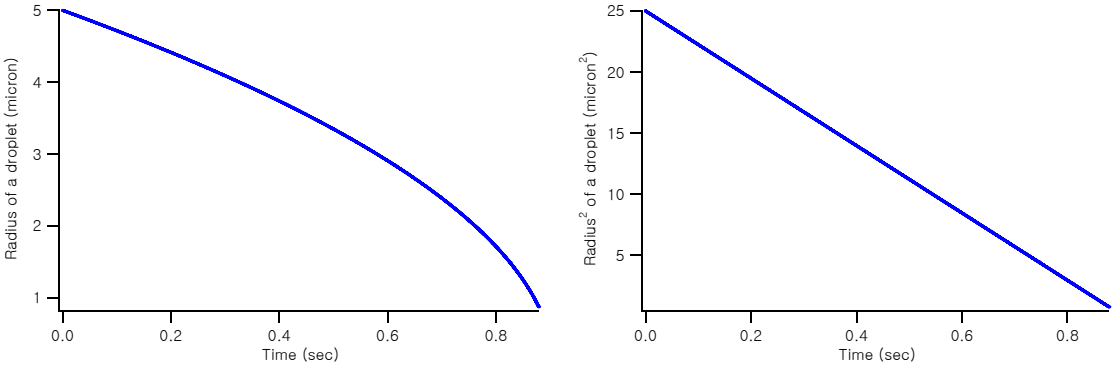


Figure 6.6 : Typical evaporation curves of water from a droplet.

The left plot shows Radius (μm) vs Time (sec), whilst the right plot shows Radius^2 (μm^2) vs Time (sec). Since the evaporation rate (μm^2/sec) is nearly a constant, we get an approximately linear decrease of Radius^2 as time evolves.

Table 6.1 : Simulation results for the evaporation of water from a droplet

Initial Radius (μm)	RH (Relative Humidity)	IR Laser* Illumination	Total Life Time (sec)	Time to reach r=1.1 μm (sec)	Time Interval (ms) for the radius of 0.9 μm ~1.1 μm	(Average) Evaporation Rate (μm^2/sec)
5	25 %	Y	0.1108	0.1055	1.75 ms	-225.7
		N	0.1110	0.1055	1.75 ms	-225.2
	50 %	Y	0.1773	0.1688	3 ms	-141.0
		N	0.1775	0.1690	3 ms	-140.7
	75 %	Y	0.3753	0.3570	6 ms	-66.6
		N	0.3770	0.3588	6 ms	-66.3
	90 %	Y	0.9628	0.9158	15.5 ms	-26.0
		N	0.9748	0.9278	15.5 ms	-25.6
	99 %	Y	8.8505	8.3723	157.5 ms	-2.8
		N	9.9398	9.4588	159 ms	-2.5

Table 6.1 continued

Initial Radius (μm)	RH (Relative Humidity)	IR Laser* Illumination	Total Life Time (sec)	Time to reach r=1.1 μm (sec)	Time Interval (ms) for the radius of 0.9 μm ~1.1 μm	Average Evaporation Rate (μm^2/sec)
10	25 %	Y	0.4420	0.4375	1.75 ms	-226.2
		N	0.4440	0.4395	1.75 ms	-225.2
	50 %	Y	0.7050	0.6965	3 ms	-141.8
		N	0.7105	0.7020	3 ms	-140.7
	75 %	Y	1.4823	1.4648	6 ms	-67.5
		N	1.5083	1.4900	6 ms	-66.3
	90 %	Y	3.7278	3.6808	15.5 ms	-26.8
		N	3.8995	3.8523	15.5 ms	-25.6
	99 %	Y	28.0345	27.556	157 ms	-3.6
		N	39.7598	39.362	159 ms	-2.5
15	25 %	Y	0.9900	0.9855	1.75 ms	-227.3
		N	0.9990	0.9938	1.75 ms	-225.2
	50 %	Y	1.5733	1.5648	3 ms	-143.0
		N	1.5988	1.5903	3 ms	-140.7
	75 %	Y	3.2755	3.2573	6 ms	-68.7
		N	3.3938	3.3755	6 ms	-66.3
	90 %	Y	8.0148	7.9678	15.5 ms	-28.1
		N	8.7740	8.7268	15.5 ms	-25.6
	99 %	Y	48.9708	48.493	157 ms	-4.6
		N	89.4597	88.979	159 ms	-2.5

6.5.3 The whole trajectory and the radius change of a droplet

Computer simulations of a droplet trajectory in a lens cube have been performed with integration of all the contents and equations in Chapter 6. There are many

parameters which affect the whole trajectory and the radius change of a droplet as shown in [Figure 6.7]. But among them, "Initial injection velocity" and "Relative Humidity" are most dominant factors.

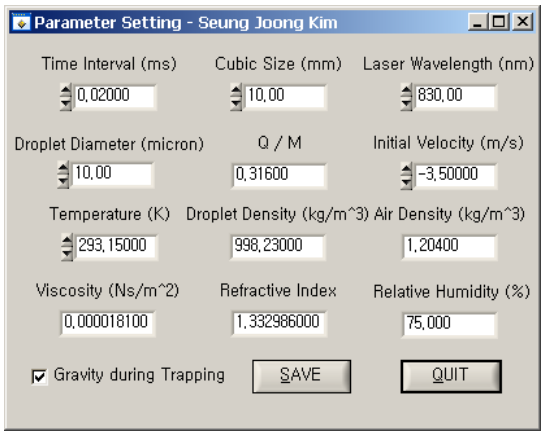


Figure 6.7 : Basic parameter setting for the simulation

Initial Injection Velocity

The "Initial injection velocity" significantly affects how far a droplet can initially fall down, until it shortly reaches to the terminal velocity in less than 3 ms.

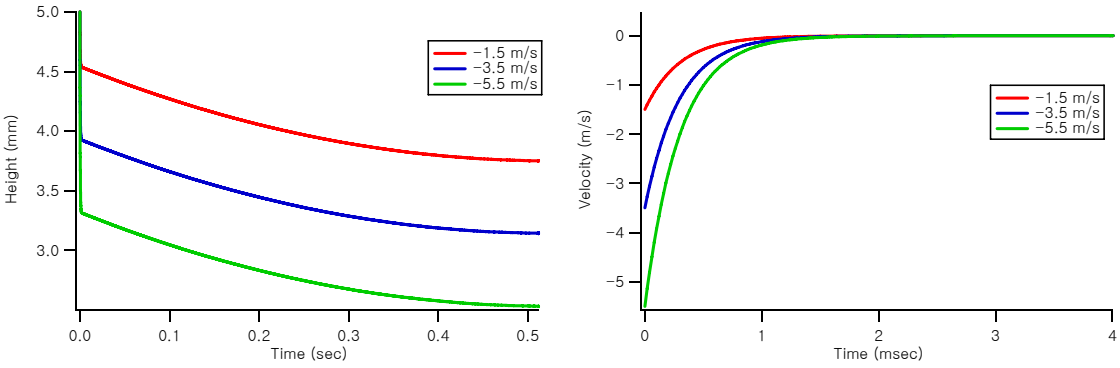


Figure 6.8 : Initial injection velocity dependence (No IR laser guidance)

The whole trajectory of a droplet (left) and the velocity change in an early time period (< 5 ms) (right). The droplet reaches its terminal velocity in a short time of less than 3 ms, in 75% relative humidity. See [Figure 6.7] for the parameter setting other than initial velocity.

Relative Humidity

On the other hand, the "Relative Humidity" mostly determines the evaporation rate and the corresponding lifetime of a droplet, as we already discussed in the previous section. Notice that the droplet comes close to the beam center (at the zero height) in 95% relative humidity, about 2.1 seconds later.

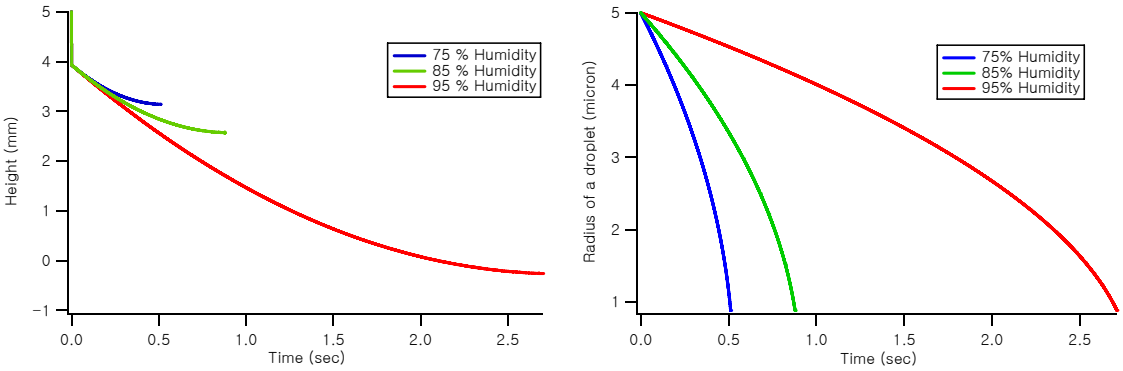


Figure 6.9 : Relative humidity dependence (No IR laser guidance)

Plots of the simulated trajectories (left) and the radius changes (right) of a droplet at different humidity. The evaporation rate decreases at higher humidity. As a result, the lifetime of a droplet significantly increases. See [Figure 6.7] for the parameter setting other than humidity.

IR laser guidance

The effects of IR laser guidance are studied for the optimal condition of the trapping. I performed three independent simulations in 95 % humidity, at the initial injection velocity of -3.5 m/s. Temperature was set at 293.15 K and the initial diameter of a droplet was 10 μm. The wavelength of the IR guiding lasers was 830 nm, with a beam power of 100 mW. At a high photon intensity setting (=14 μm beam waist), as soon as a droplet enters the IR guiding laser beam, the droplet starts to be trapped around the beam center until it completely evaporates. In a case of low photon intensity setting (=40 μm beam waist), we could not find any noticeable effects by the IR guiding lasers. Both of the trajectory and the radius changes look identical to the case of non-IR laser guidance. In conclusion, the intensive IR laser focusing (about 14 μm beam waist) is required for the efficient IR laser guidance.

While a droplet is illuminated by the IR guiding lasers of high intensity, the droplet absorbs the heat energy from the IR laser beams, resulting in a higher evaporation rate. Thus the droplet trapped by the high intensity IR lasers evaporates faster than others.

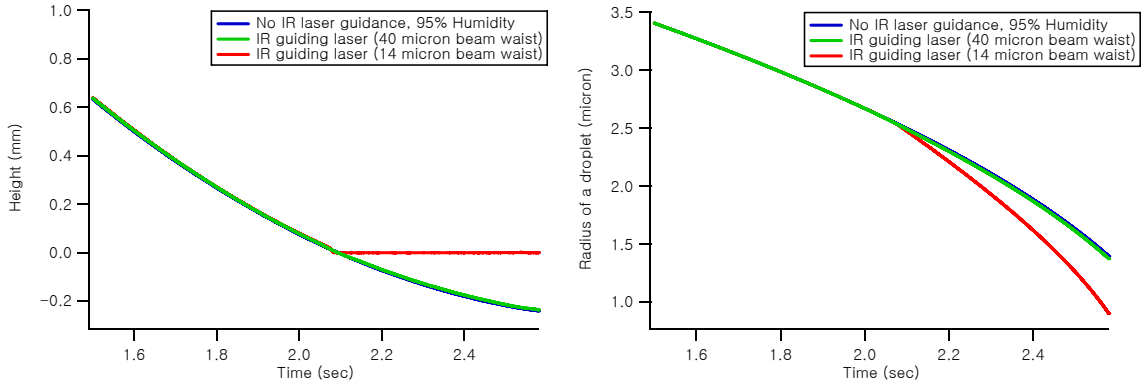


Figure 6.10 : Simulation results for IR laser guidance

Plots of the simulated trajectories (left) and the radius changes (right) of a droplet with different 830 nm IR laser guidance settings. See [Figure 6.7] for the basic parameter setting other than humidity (The humidity was fixed at 95%.)

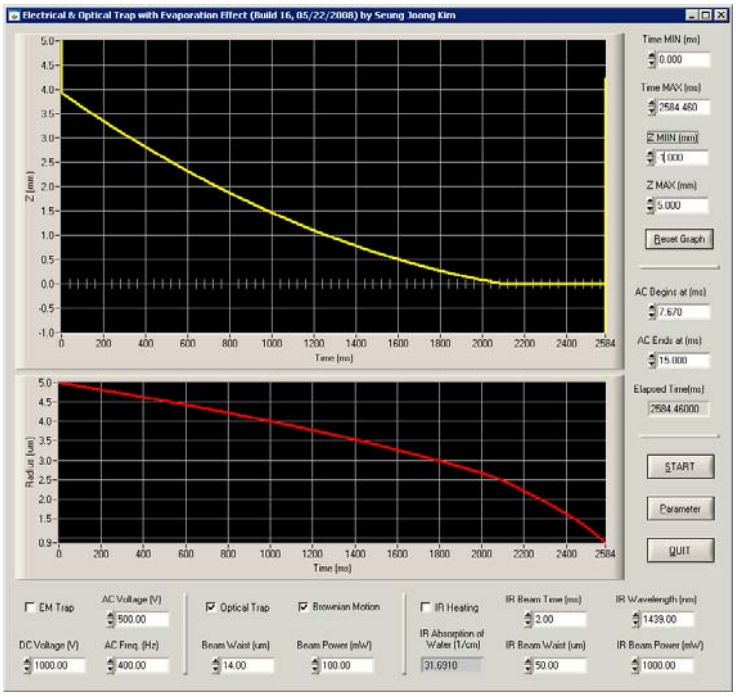


Figure 6.11 : The whole trajectory of a droplet, by IR laser guidance of high intensity

A screen capture from the simulation program, written in C language under NI Labwindows/CVI 8.5 GUI environments. Two of 830 nm IR lasers are used for the laser guidance at 95 % humidity, with a high intensity (100 mW beam power, 14 μm beam waist). See [Figure 6.7] for the basic parameter setting.

An alternative way – utilizing low humidity and an IR heating pulse for the fast evaporation

In addition, there is an alternative way for the trapping of a droplet by utilizing a high evaporation rate in a low humidity. This is just an opposite way from what we have done so far. By delicately tweaking the humidity, the initial injection velocity, and an IR heating pulse, we may trap a droplet even without any help of guiding lasers for a significant amount of time.

For example, if we set a low humidity of less than 1 %, the evaporation rate is extremely high so that we get a short lifetime (< 80 ms) for a droplet of 10 μm diameter. But remember, as the droplet becomes smaller, its terminal velocity comes close to zero – so eventually the droplet stops for a moment [Figure 6.12]. With an aid of an IR heating pulse (1430 nm wavelength) from the bottom, we can actively control the evaporation rate and the whole trajectory of a droplet. (Water is known to absorb 1000 times more at 1430 nm than at 830 nm, see [Figure 6.2].) The IR heating pulse should be focused onto the tip of the droplet generator, so that it can be easily aligned.

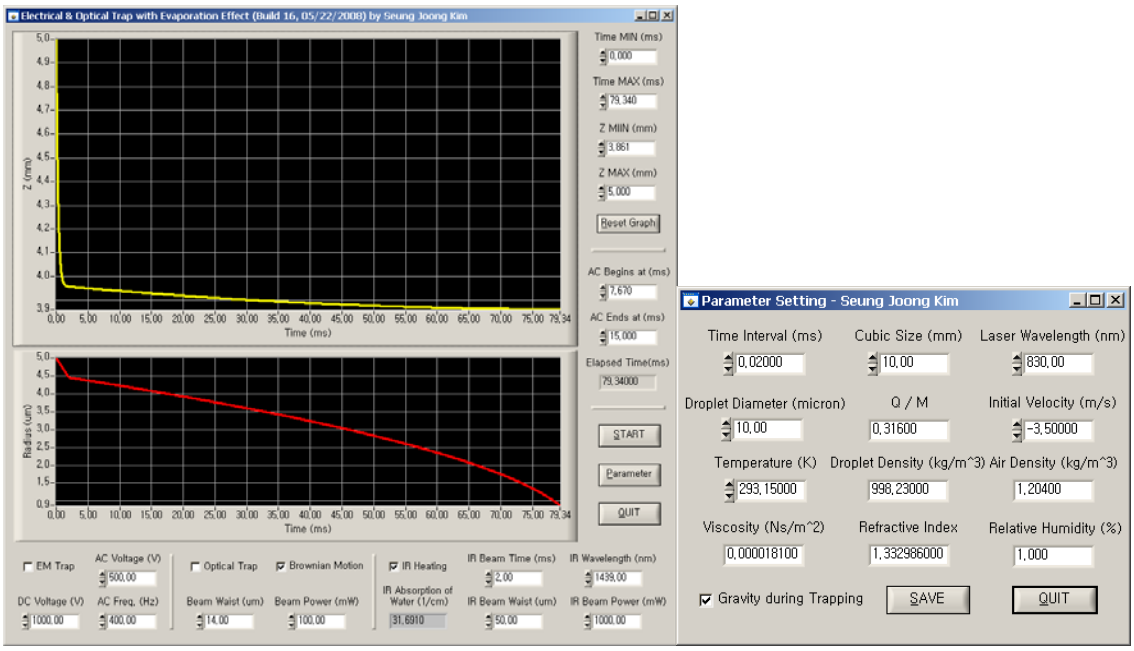


Figure 6.12 : The whole trajectory of a fast evaporating droplet

A screen capture from the simulation program. The IR heating pulse of 1430 nm wavelength was applied for 2 ms with a high power of 1000 mW, for the fast evaporation in 1 % humidity.

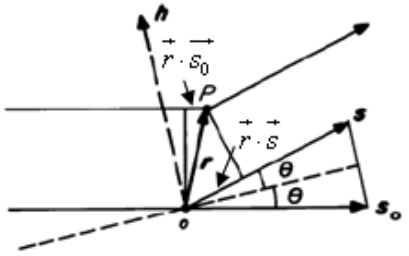
Appendix A Description of small angle X-ray scattering experiments

A.1 Background theory

By analyzing X-ray scattering profiles at a small angle (typically $2\theta < 1^{\circ}$), we can directly figure out how big the particle is. We use the radius of gyration R_g to determine the size of the protein [12, 67], which is defined as the following

$$R_g^2 = \frac{\int_V s^2 \rho(s) d^3 s}{\int_V \rho(s) d^3 s}$$

where $\rho(s)$ is the density of a sub-volume of the particle located at a position vector s from the center of mass. Indeed, the radius of gyration is defined as the root mean square of mass-weighted distances of all sub-volumes in a particle, from the center of mass.



Scattering by two point centres.

Analyzing SAXS data, we can calculate the radius of gyration easily through the Guinier Plot [13] as follows. By definition, the scattering vector is a difference vector of the scattered beam unit vector $\vec{s}$ and the transmitted beam unit vector $\vec{s_0}$.

$$\vec{q} \equiv \frac{2\pi}{\lambda}\left(\vec{s} - \vec{s_0}\right)$$

The optical path difference is defined as $\Delta l = \vec{r} \cdot \left(\vec{s} - \vec{s_0} \right)$, and the phase difference is defined as $\varphi = 2\pi \frac{\Delta l}{\lambda} = \frac{2\pi}{\lambda} \vec{r} \cdot \left(\vec{s} - \vec{s_0} \right) \equiv \vec{q} \cdot \vec{r}$.

Since $\left| \vec{s} - \vec{s_0} \right| = 2 \sin\theta$, as shown from the above figure, the scattering vector magnitude is directly proportional to $\sin\theta$,

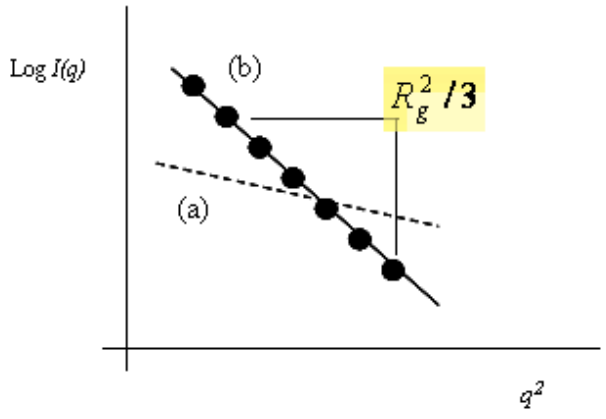
$$\left| \vec{q} \right| = \frac{4\pi n \sin\theta}{\lambda_o}$$

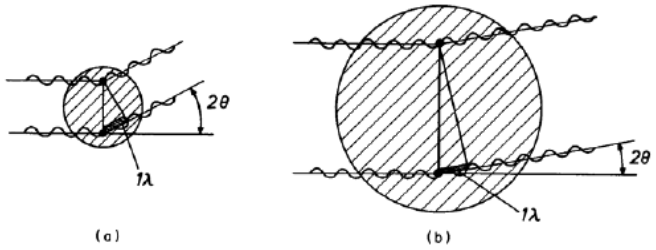
where $\lambda = \frac{\lambda_0}{n}$

By Guinier approximation as described in many references [12, 67], the scattering intensity at the scattering vector q, I(q) can be expressed as,

$$\ln I(q) \simeq \ln I(0) - \frac{q^2 R_g^2}{3}$$

It is valid only when $qR_g < 1.3$ for spherical objects. Therefore at the small angle satisfying $qR_g < 1.3$ for spherical objects, the slope of the SAXS profile in the Guinier plot (q^2 *vs* ln I(q)) should be directly proportional to the square of the radius of gyration, R_g^2.





For example, a small particle tends to scatter more, so the X-ray photons will be spread more widely, resulting in low beam intensity at the small angle region. Thus we can get a low slope (a) in the Guinier plot. On the other hand, a large particle doesn't scatter much, so the X-ray photons will be more densely packed at the small angle region, resulting in a high slope (b).

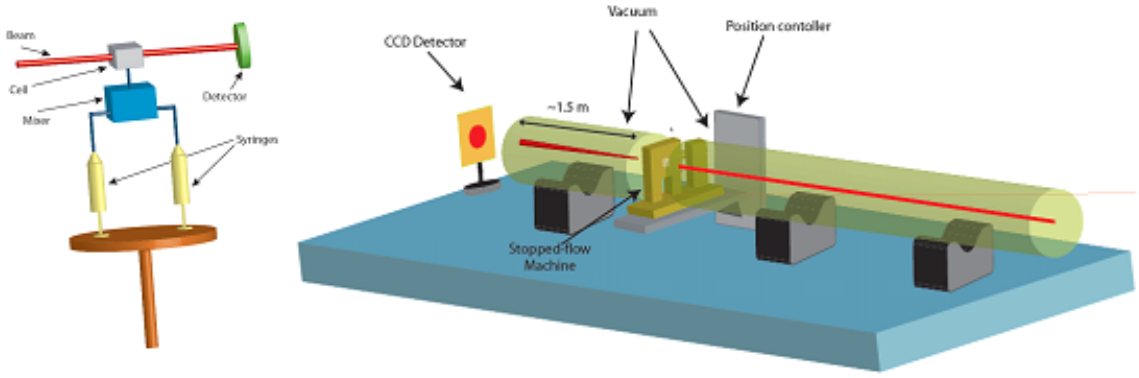
A.2 Experimental setup at Argonne National Laboratory

The SAXS experiment was performed at the BioCAT-18 section of the Advanced Photon Source (APS) at Argonne National Laboratory, in collaboration with Charles Dumont (a Ph.D. student in Physics), the BioCAT-18 group, and the Kihara group in Kansai Medical University, Japan.

For the protein folding kinetics, the stopped-flow apparatus (Unisoku, Japan) was installed for fast mixing to dilute a denatured protein solution within a dead time of less than 5 ms, so that the following protein collapse (kinetics upon refolding) could be monitored in combination with SAXS. Despite using the brightest X-ray source in the world, exposure times of more than 100 ms are required for the collection of reliable data, which is quite a large time scale compared with the sub-ms folding time scale of $\lambda_{6\text{-}85}$*. Sub-zero temperature (-28 °C) and high viscosity solvent (45 % Ethylene Glycol / 55 % water by volume, having ~10 times higher viscosity than water at 298K) slow down the kinetics enough for the measurement to be possible. [162]

Our stopped-flow apparatus has an instrument dead time of less than 5 ms with a mixing ratio of 1 to 6. For the fast mixing, a piston is pushed by a nitrogen gas controlled mechanical system, moving each solution into the mixer with a volume ratio of 1 to 6 (for

example, one volume of denatured protein in 5 M Guanidine Hydrochloride vs. 6 volumes of 0 M Guanidine Hydrochloride buffer). The instantly-mixed protein solution (in 0.7 M Guanidine Hydrochloride) flows through an observation window cell (made of sapphire) for the SAXS measurement. We use a timing box to integrate the x-ray scattering for a specific time window.[29] For the details of timing box setup, refer to Appendix A in reference [29].



Schematic diagram of Stopped-Flow apparatus (left), Experimental Setup at Argonne Lab. (right). From ref. [29]: Larios, E., *a Computational-Experimental Study of Small Globular Proteins*, in *Physics Ph.D. Thesis*. 2005, University of Illinois at Urbana-Champaign.

A.3 How to design and perform a solution X-ray scattering experiment

This manual explains how to design and perform a solution x-ray scattering experiment at the BioCAT-18 section of the Advanced Photon Source at Argonne National Laboratory, and I attach this for completeness. Most of the contents in this section are based on the published web documents in Argonne National Laboratory website [163], (cited from http://www.bio.aps.anl.gov/techniques/SAXS-HOWTO.html) and it delivers quite useful information especially for the beginner.

A.3.1 Range of Q (scattering vector magnitude)

We typically do Small Angle X-ray Scattering (SAXS) or Wide Angle X-ray Scattering (WAXS) experiments at an x-ray photon energy of 12 keV. The Aviex CCD

detector that will be used for SAXS or WAXS measurements has an active area of approximately 160 x 80 mm^2. The zero-order beam stop---after masking---typically has an effective diameter of 4.6 mm. We can set a sample-to-detector distance for SAXS experiments at ~950, ~1400, ~1800, ~2300, or ~2750 mm, which corresponds to a Q range, when offsetting the beam stop 60 mm from the detector center, of 0.015-0.9 $Å^{-1}$, 0.01-0.6 $Å^{-1}$, 0.008-0.47 $Å^{-1}$, 0.006-0.37 $Å^{-1}$, and 0.005-0.3 $Å^{-1}$, respectively. Switching between different Q ranges can take between four and eight hours.

For the WAXS instrument, the beam stop is set at the center of the detector and the sample-to-detector distance is fixed at 180 mm. This corresponds to a Q range of 0.08-2.5 $Å^{-1}$.

A.3.2 Reducing radiation damage

Longer exposure to high power X-ray beams will lead to radiation damage to the protein. Protein aggregation also can result from radiation damage. To reduce radiation damage, we can shorten the exposure time (down to several hundred milliseconds), and lower the sample temperature (down to -28 °C), and even add small amounts (~100 mM) of cryoprotectants, such as glycerol, ethylene glycol and sucrose.[164] But the ideal condition must be determined by trying various experimental parameters prior to collecting main SAXS data.

A.3.3 Sample concentration

For SAXS

For proteins of a size comparable to lysozyme or cytochrome c, a concentration of 2 mg/ml can give reasonably good data quality. If the protein has twice the size of lysozyme or cytochrome c, the concentration can be reduced by a factor of two. Higher concentrations can be used to give better data quality if the protein does not suffer from aggregation. We can measure samples with a concentration of 0.5 mg/ml at long

exposures and have good data quality in the low q region.

DNA and RNA scatter x-rays more strongly than proteins, so the required concentration can be about 5 fold lower than proteins.

For WAXS

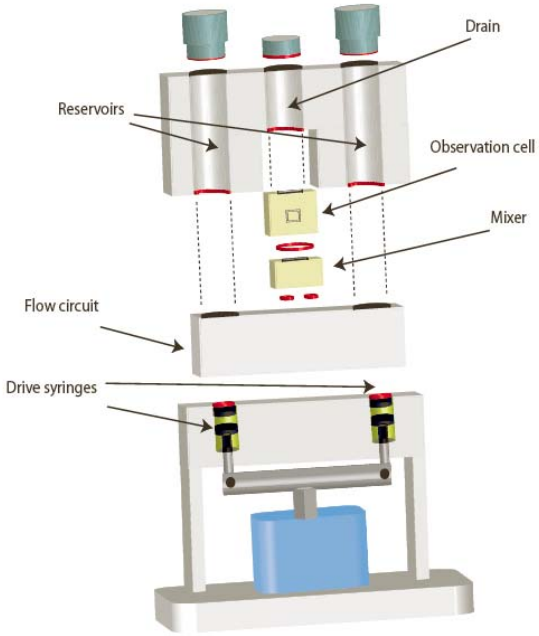
Here we acquire data in the intermediate to high q region (0.05-2 $Å^{-1}$) where aggregation of sample molecules have little effect in the data. In this case we can use much higher concentrations than for SAXS in order to increase the weak scattering signal in the high q region. The concentration can be 5 or even more times higher than for SAXS. If you also need SAXS data, you can dilute WAXS samples and use the diluted samples for SAXS measurements if aggregates break up easily upon dilution. If dilution does not break up aggregates or it takes a very long time to break up aggregates, prepare separate SAXS samples at the desired low concentrations.

Buffer Solution

Scattering data taken on a protein solution contains signals from both the protein and the buffer. Scattering measurements should be done on both the protein solution and the buffer. The scattering signal from the buffer alone is then subtracted from the solution scattering in order to get the scattering signal of only the protein. The buffer for measurement must match that in the protein solution.

One way to get matched buffers is to dialyze the protein solution in a buffer for a certain time, and then bring both the protein solution and the buffer for SAXS measurement. It is recommended to bring plenty of buffer (> 1 L).

A.3.4 Loading samples



From ref. [29]: Larios, E., *a Computational-Experimental Study of Small Globular Proteins*, in *Physics Ph.D. Thesis*. 2005, University of Illinois at Urbana-Champaign.

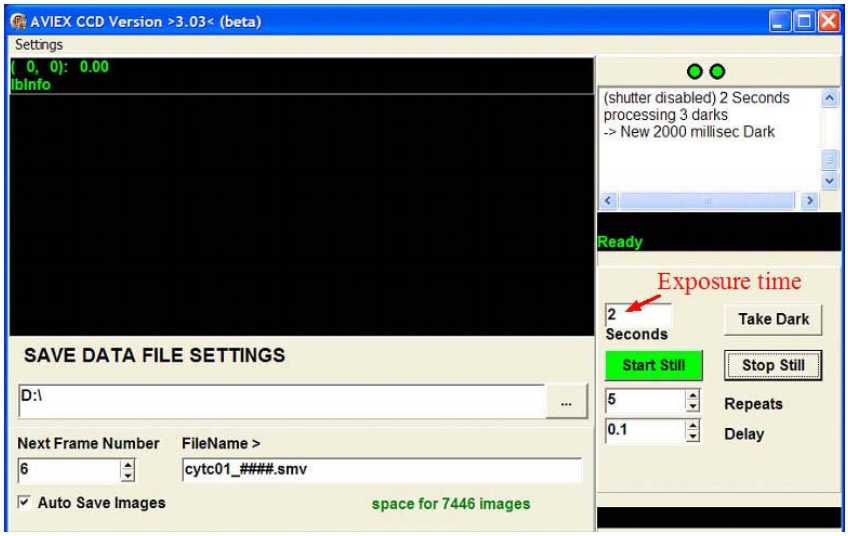
The protein solution and buffer are contained in reservoirs before being loaded into the observation cell. We need to filter samples and centrifuge the protein solution and buffer to remove any bubbles inside solutions before loading. The protein solution is measured immediately following the measurement of its matched buffer. Before switching to the next buffer and protein solution, it is recommended to flush the observation cell with the sequence: water, 20% bleach, water, 100% ethanol, water, acetone, and water to remove any possible protein deposits left on the observation cell wall by interactions with the high-flux x-ray beam.

For the operating principle of the stopped flow instruments in detail, refer to Appendix A of Ref. [29] : a Ph.D. dissertation by Edgar Larios.

A.3.5 Acquiring data

The client program which controls the Aviex CCD currently resides on the computer named "Godzilla." Double-click the icon "Shortcut to Aviex CCD" to start the

program. This opens the following window:



From ref. [163]: The published web documents in Argonne National Laboratory website by BioCAT-18

First, have a directory created for you on the computer and select this working directory by clicking on the "..." button. Name your experiment file in the format name_####.smv, where #### is the index that will be filled in automatically by the client program with the number you specify in the "Next Frame Number" box. This number will automatically increment by 1 each time a frame of exposure is taken. You may change it at any time.

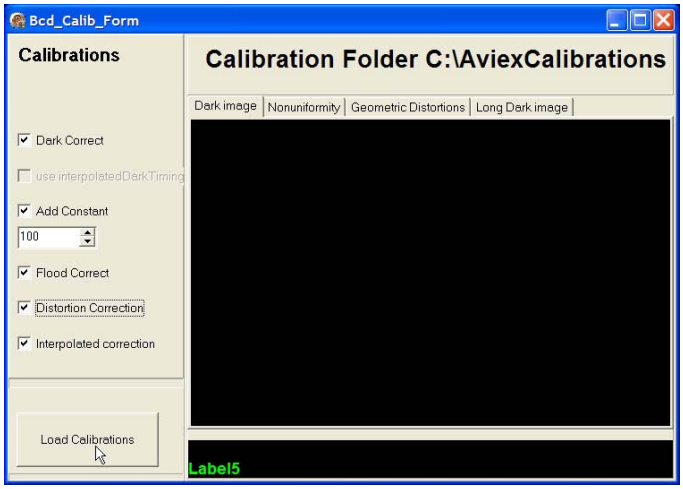
It may be convenient to name a protein and its buffer with different names such as proA_####.smv and bufferA_####.smv. Do not use a numeral as the first letter of the filename, since the Igor Pro program we will use later for data reduction does not like this. Also, keep the total length of the filename shorter than 20 characters.

Check the "Auto Save Images" box to have your images automatically saved - unless you do not wish to save them (e.g. during a practice run). Select your desired exposure time for each frame of the CCD image.

Every time you change exposure times, you need to record a dark image for

background subtraction. To do this,

1. Close the Normally Open shutter with the flip-switch on the XIA control box.
2. Click on "Take Dark" in the Aviex client program
3. Save the dark image as dark.smv in the directory C:\Aviex Calibrations (you will be prompted for this information)
4. Open the "Configure" dialog by selecting it under the "Settings" menu on the Aviex client program:
5. Make sure all the boxes on the left are selected.
6. Click "Load Calibrations"



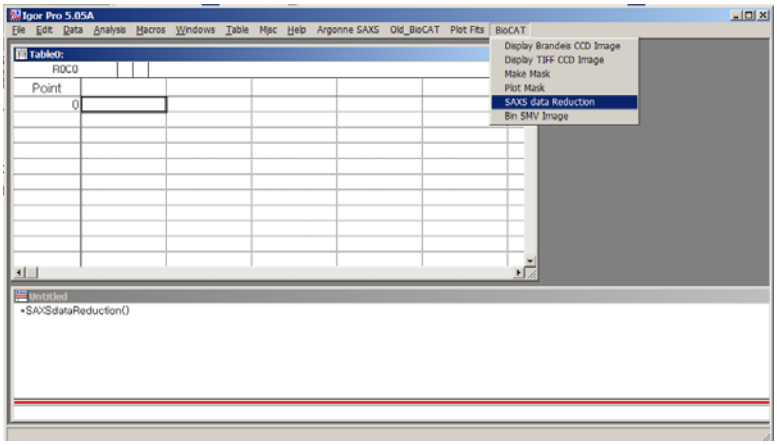
From ref. [163]: The published web documents in Argonne National Laboratory website by BioCAT-18

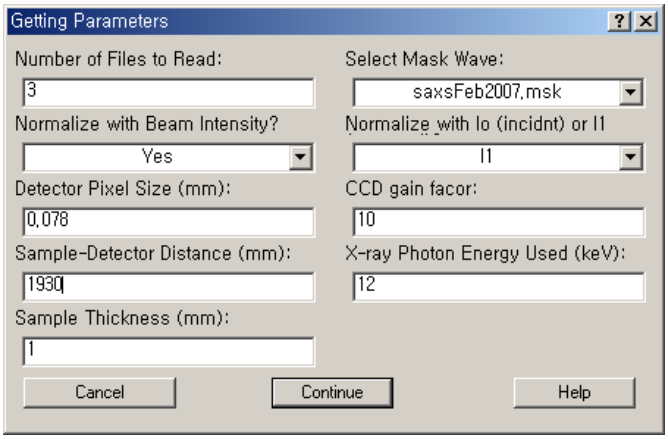
Experimenters are encouraged to take darks periodically as the detector can drift slightly over the course of a few hours. You may minimize the calibration window if you like. Go back to the client program to select the number of continuous repeats of frames (exposures) you desire each time you click on the "Start Still" button and choose 0.1 or 0.2 sec for the "Delay" between each exposure. Even if you take only one exposure, the delay time should still be set. If you have started a continuous run of several exposures accidently, you can press the "Stop Still" button to interrupt the run.

A.3.6 Data reduction

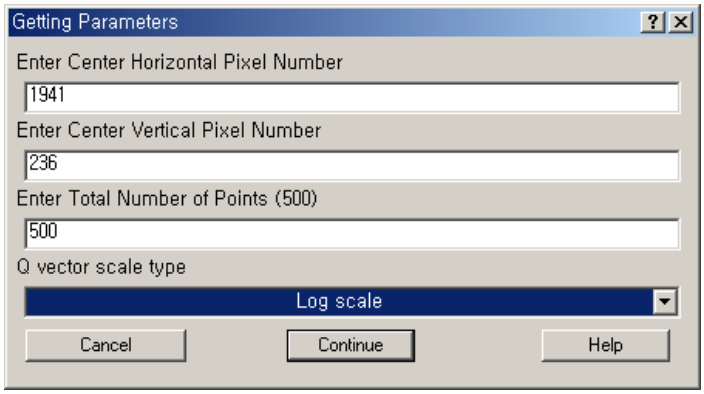
The acquired SAXS/WAXS data are saved as CCD image files and they need to be integrated azimuthally around the beam center to generate data of scattering intensity versus scattering vector (I vs. Q). The program Igor Pro will be used to do data reduction of the SAXS/WAXS images. We need to install additional user Igor Pro procedures written by BioCAT-18.

First, map the folder on Godzilla where the SAXS/WAXS images are stored onto a local network folder on your computer. Then open the Igor Pro program by double clicking on the Igor Pro icon, and choose "BioCAT → SAXS data Reduction" to input the necessary SAXS parameters:





Fill in the number of frames to be processed that have the same base name filename_####.smv which is usually used for the same protein sample or buffer. Select the mask image that the BioCAT staff has created for you. You will need to normalize your data with beam intensity, so select "transmitted intensity I1" for normalization. The sample thickness is the diameter of the capillary tube. The detector pixel size is 0.039 mm without binning and 0.078 mm for 2x2 binning, depending on which you chose. Fill in the x-ray energy and the sample-to-detector distance values. Normally you do not need to "Sum All Files to a Single Frame." After filling in all the needed parameters, click on the "Continue" button to open up the next dialogue box:



Fill in the horizontal and vertical pixel numbers of the beam center. Select the total number of data points you would like to display on a plot and the Q vector scaling mode. It should be noted that the more data points you would like to generate, the less area of the CCD image is used to average into the individual data points and, hence, higher statistical errors and less smooth experimental curves are obtained. On the other hand, if too few points are used, you generate data more spatially smeared and loose spatial resolution. It is really up to your needs to select the proper number of data points.

When all parameters are filled in, click on the "Continue" button to let Igor do the data reduction job. It will take a while to reduce a few dozen SAXS/WAXS images. For each scattering image Igor Pro returns three columns of reduced data: the Q vector data with "q" prepended to the filename, the intensity data with an "r" prepended, and the intensity error data with an "s" prepended. For example, one would get the files

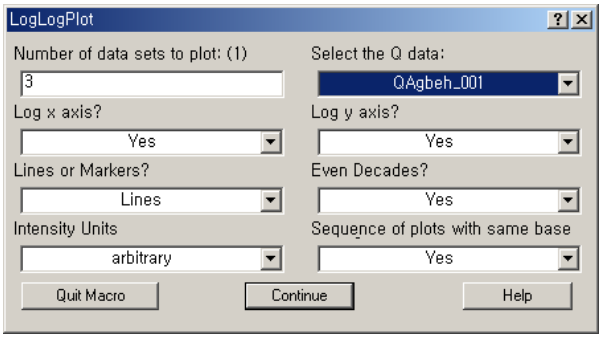
qproteinA.0001, rproteinA.0001, and sproteinA.0001 for the image proteinA_0001.smv. Note that in the reduced data, the image extension ".smv" is removed.

A.3.7 Analyzing data and the Guinier plot

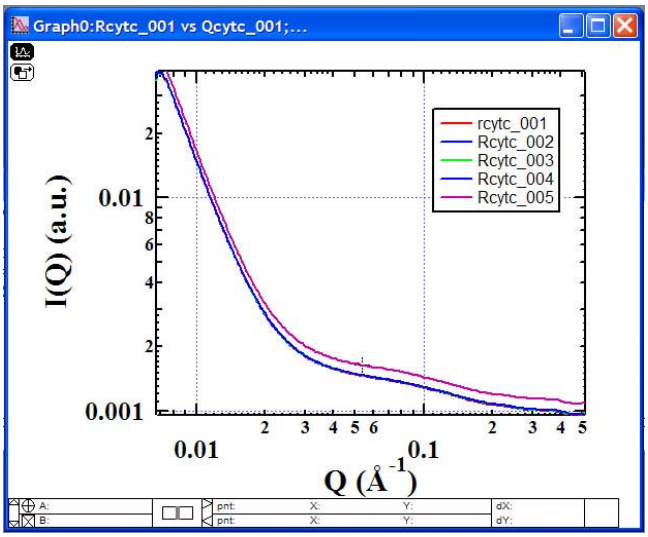
Log-Log Plot

Macros have been written in Igor Pro to allow you to do quick data analysis on-site. First, we want to examine the scattering pattern of the protein in solution and check for protein aggregates in the solution.

In Igor Pro, click on tab "Plot Fits → LogLog Plots → LogLogPlot" to open the following dialogue window:



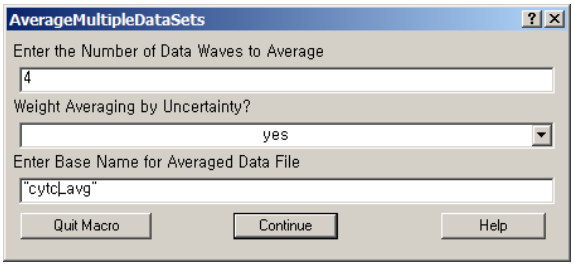
We want to display all the reduced data for the same protein sample on a log-log scale plot to compare all the curves. Select the first frame number in the sequence and check to plot a sequence of frames with the same base name (i.e. for all the frames for the same sample). We want to plot all the curves as lines for now so that they are easier to compare with each other (the purpose for doing this is to identify the few curves that deviate from the average so that we will remove them). Click on the "Continue" button to make a plot similar to the following:



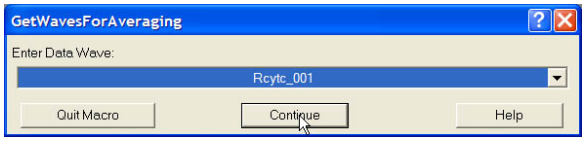
From ref. [163]: The published web documents in Argonne National Laboratory website by BioCAT-18

Averaging data

Notice that frame 5 deviates from the other curves. Therefore, when we do averaging, we use only curves 1 to 4. To perform data averaging, click on tab "Plot Fits → Modify Data → Average Multiple Data Sets" to open the following dialogue box:



Fill in the number of frames to be averaged (in this case, 4) and use "weight averaging by uncertainty" (the average is weighted by the data error bars, the bigger the errors the less the weighting). Give a name for the average and click on the "Continue" button to bring up the following dialogue box for choosing the data to be averaged:

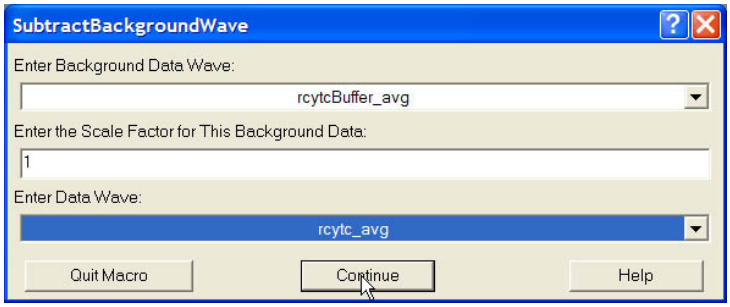


From ref. [163]: The published web documents in Argonne National Laboratory website by BioCAT-18

Each frame must be selected individually from the pull-down menu. Click "Continue" to proceed to the next frame. The program will calculate the averaged data when all the needed frames have been selected. If you have done 20 frames for a particular protein sample, you will find that selecting 20 frames one by one is tedious. There exist some Igor shortcuts which BioCAT staff will alert you to.

Background Subtraction

Having averaged the data for both the protein solution and its buffer, we now subtract the buffer scattering from the protein solution scattering. To do this, click on the tab "Plot Fits → Modify Data → Subtract Background Wave" to open the following dialogue box:



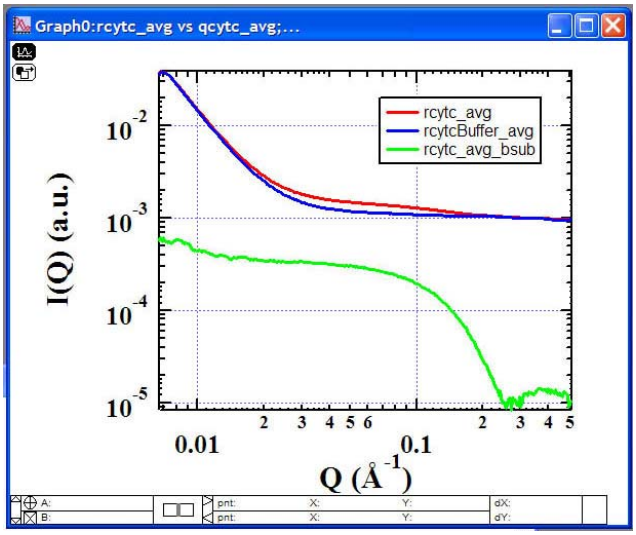
From ref. [163]: The published web documents in Argonne National Laboratory website by BioCAT-18

Enter the averaged scattering data from the buffer as the background data, and the averaged data from the protein solution as the Data to be processed. The "Scale Factor" is the fraction of the buffer scattering that you want to remove. Since most protein solutions are at very low concentrations, the volume content of the buffer in the protein solution is almost 100%, so "1" can be used for the "Scale Factor".

What if you have a protein solution at 10 mg/ml? Suppose the protein has a density of 1.4 g/cc, 10 mg/ml is about 0.71% protein by volume in the solution. Thus, the buffer has a volume fraction of 0.993 in the solution. In this case, you may use 0.993 for the "Scale Factor." However, it introduces additional error since the scattering from the buffer also contains the scattering from the observation cell and this process only subtracts 0.993 of the observation cell scattering. To correct this error, it is best to do a

scattering measurement on the empty observation cell as well and subtract the observation cell scattering by 100% from both buffer scattering and protein solution scattering. Then use the scale factor 0.993 for the net buffer scattering and the net protein solution scattering.

Now you can plot the averaged data and the background removed data (with _bsub attached to the data filename) on the same plot (recall above procedures to make a plot. But you have to select each data set one by one, since they do not have the same base name now):



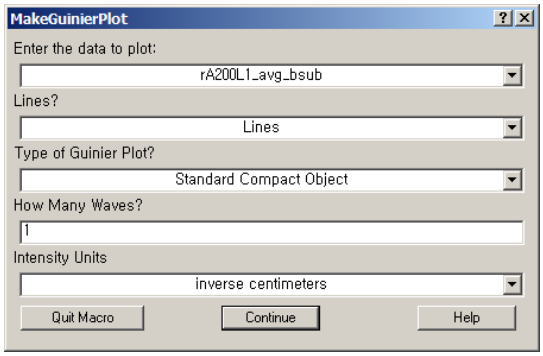
From ref. [163]: The published web documents in Argonne National Laboratory website by BioCAT-18

In this plot, the red and blue curves are for the averaged protein solution data and the averaged buffer data, respectively; and the green curve is for the protein solution with buffer background removed. The slight bending up of the green curve at the very low Q region indicates the presence of a small amount of aggregates (aggregate-free protein would show a flat scattering curve in the low Q region). In rare cases, you will see a higher scattering signal of the blue buffer curve than the red protein solution curve. This is the outcome of some errors during the measurements. It could be that your buffer does not match the one in the protein solution or that, at some point during measurement, the beam moved. You should repeat the experiment. Should you get similar results, some investigation will be required to figure out what is happening.

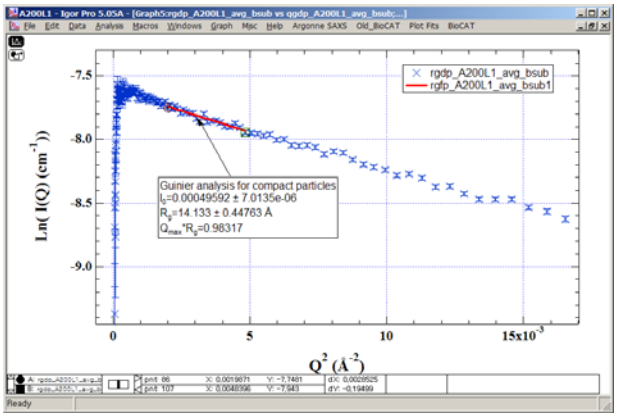
First, check your preparation of protein solutions and buffers. It is also possible that, for concentrated protein solutions, the buffer may have higher signal than the protein solution. After you have removed the empty cell scattering as discussed above and used the correct "scale factor" (less than 1) for background subtraction, you will end up with the correct data for the net protein scattering.

The Guinier Plot

In Igor Pro, click on tab "Plot Fits → Special Plots → Make Guinier Plot" to open the following dialogue window:



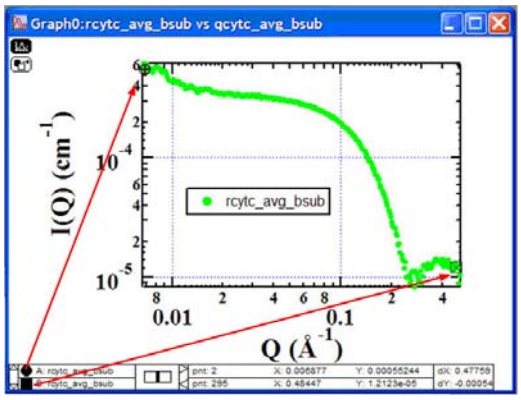
Entering appropriate parameters and clicking "continue" button, we can easily generate the Guinier plot. Also if you click on tab "Plot Fits → Special Plots → Perform Guinier Fit", we can perform a Guinier fit to calculate the radius of gyration as follows. You need to select a range by dragging cursor "A" and "B" to perform a Guinier fit.



The Guinier plot and a corresponding Guinier fit to calculate the radius of gyration

A.3.8 Selecting a range of data and saving as an ASCII file

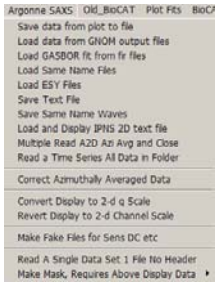
Now that we have reduced the CCD image into 1D data and have removed the background scattering, we wish to output the data as an ASCII file so that it may be used by some other analysis programs such as GNOM and CRYSOL.[38] As shown below, on the plot with the data to be output, drag cursor "A" and "B" with the left mouse button to the beginning and end points of the region of data that you wish to output.



From ref. [163]: The published web documents in Argonne National Laboratory website by BioCAT-18

The Save Data menu

Cursor "A" must be the beginning point and cursor "B" the end point. Then click on the tab "Argonne SAXS → Save data from plot to file" to save the data to a folder that you select. You will notice that 5 rows down, the tab you just clicked has the option "save text file." You can use that to save the whole range of data. The first column of the saved data is the Q vector, the second column contains the scattering intensity, and the third column is the error of the scattering intensity.



The drop down menu

A.4 Packing list for SAXS experiments in Argonne National Laboratory

These are typical lists we need to bring into Argonne National Laboratory.

- Protein Sample (normally lyophilized already), in a Styrofoam box with plenty of frozen ice packs.
- The stopped-flow instrument with temperature gauge and observation cells
- Extra battery for temperature gauge
- A chiller, but we need to wash it before leaving.
- Coolant for the chiller (Ethylene Glycol + Methanol Mixture, 4 gallon)
- Empty containers for the disposal of coolant after experiments.

- Phosphate Buffer (50 mM Phosphate, pH = 7.0)
- Mixing Buffer (50 mM Phosphate, 45% Ethylene Glycol by volume, pH = 7.0)
- Protein Buffer (50 mM Phosphate, 45% Ethylene Glycol by volume, 5 M Guanidine Hydrochloride, pH = 7.0)
- Chemicals: Guanidine Hydrochloride, Ethylene Glycol, Methanol, Sodium Phosphate, Sodium Chloride, and etc.
- Plenty of Pipets and Tips
- Plastic transfer pipets
- Syringe + Filters (> 30 pieces)
- Amicon (Small / Big), with 3 kDa and 30 kDa membranes.
- Plastic Tubes (50 mL / 14 mL)

- Timing Box
- Don’t forget to bring Badge / ID card for APS
- Peristaltic Pump
- Notebook Computer
- Calculator
- DVD Blank Media / External Hard Disk Drive
- Rent a car (van) from university carpool.

Appendix B Biochemical protocols

B.1 Protein sequences and basic characteristics

For the calculation of Molecular Weight, Extinction Coefficient, etc, the web-based Peptide Property Calculators are used (Northwestern university: http://www.basic.northwestern.edu/biotools/proteincalc.html and INNOVAGEN: http://www.innovagen.se/custom-peptide-synthesis/peptide-property-calculator/peptide-property-calculator.asp).

B.1.1 Lambda repressor (λ^{*}_{6-85})

Pseudo Wild Type (Y22W)

< Amino acid sequence in 1 letter code >

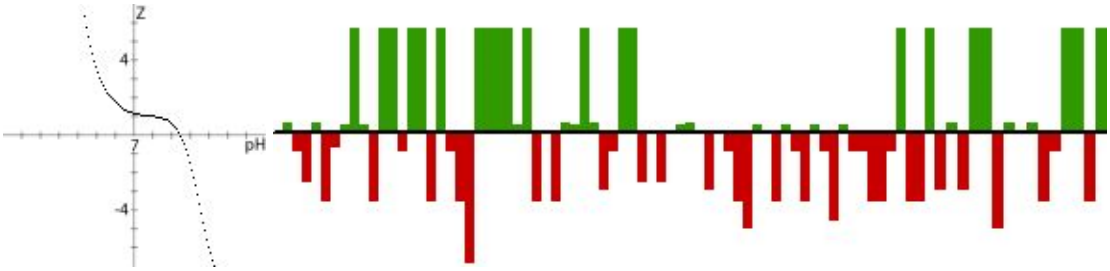
GSHMSLTQEQLEDARRLKAI**W**EKKKNELGLSQESVADKMGMGQSGVGALFNGI
NALNAYNAALLAKILKVSVEEFSPSIAREIR

< Amino acid sequence in 3 letter code >

Gly-Ser-His-Met-Ser-Leu-Thr-Gln-Glu-Gln-Leu-Glu-Asp-Ala-Arg-Arg-Leu-Lys-Ala-Ile-**Trp**-Glu-Lys-Lys-Lys-Asn-Glu-Leu-Gly-Leu-Ser-Gln-Glu-Ser-Val-Ala-Asp-Lys-Met-Gly-Met-Gly-Gln-Ser-Gly-Val-Gly-Ala-Leu-Phe-Asn-Gly-Ile-Asn-Ala-Leu-Asn-Ala-Tyr-Asn-Ala-Ala-Leu-Leu-Ala-Lys-Ile-Leu-Lys-Val-Ser-Val-Glu-Glu-Phe-Ser-Pro-Ser-Ile-Ala-Arg-Glu-Ile-Arg

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
84	9159.5	6970	11082

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
1.1	9.4	0.2	45 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

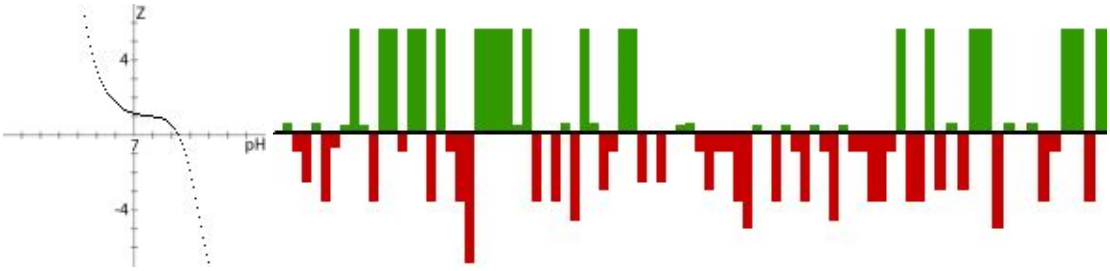
Y22W Q33Y G46A G48A

< Amino acid sequence in 1 letter code >

GSHMSLTQEQLEDARRLKAI**W**EKKKNELGLS**Y**ESVADKMGMGQS**A**V**A**ALFNGI NALNAYNAALLAKILKVSVEEFSPSIAREIR

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
84	9222.6	8250	11159

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
1.1	9.3	0.1	44 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

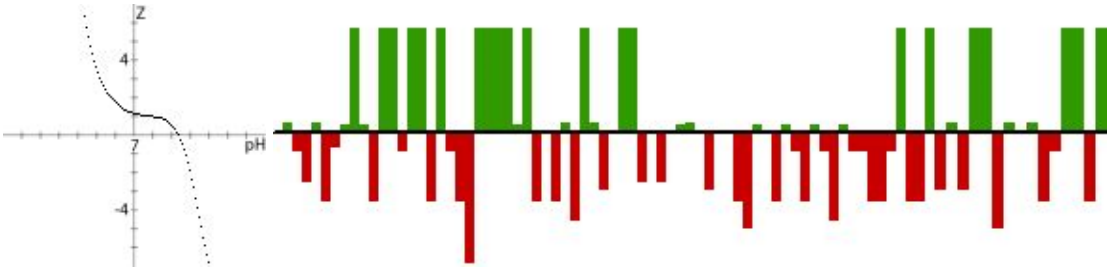
Y22W Q33Y A37G A49G

< Amino acid sequence in 1 letter code >

GSHMSLTQEQLEDARRLKAI**W**EKKKNELGLS**Y**ESV**G**DKMGMGQSGVG**G**LFNGI
NALNAYNAALLAKILKVSVEEFSPSIAREIR

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
84	9166.5	8250	11091

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
1.1	9.3	0.1	44 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

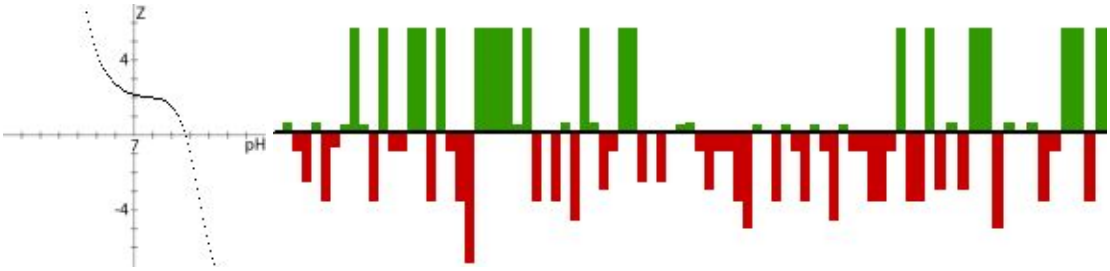
D14A Y22W Q33Y G46A G48A

< Amino acid sequence in 1 letter code >

GSHMSLTQEQLE**A**ARRLKAI**W**EKKKNELGLS**Y**ESVADKMGMGQS**A**V**A**ALFNGI
NALNAYNAALLAKILKVSVEEFSPSIAREIR

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
84	9178.6	8250	11105

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
2.1	9.7	0.1	43 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

B.1.2 *fyn*-SH3 wild type (with a His-tag)

< Amino acid sequence in 1 letter code >

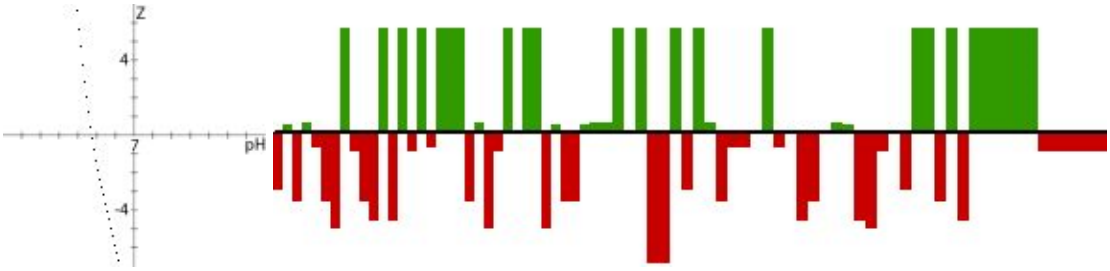
VQISTLFEALYDYEARTEDDLSFHKGEKFQILNSSEGDWWEVRSLTTGETGYIPSN
YFAPVDRLDYKDDDDKHHHHHH

< Amino acid sequence in 3 letter code >

Val-Gln-Ile-Ser-Thr-Leu-Phe-Glu-Ala-Leu-Tyr-Asp-Tyr-Glu-Ala-Arg-Thr-Glu-Asp-Asp-Leu-Ser-Phe-His-Lys-Gly-Glu-Lys-Phe-Gln-Ile-Leu-Asn-Ser-Ser-Glu-Gly-Asp-Trp-Trp-Glu-Val-Arg-Ser-Leu-Thr-Thr-Gly-Glu-Thr-Gly-Tyr-Ile-Pro-Ser-Asn-Tyr-Phe-Ala-Pro-Val-Asp-Arg-Leu-Asp-Tyr-Lys-Asp-Asp-Asp-Asp-Lys-His-His-His-His-His-His

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
78	9254.9	17780	11199

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
-9.4	4.7	0.2	44 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

B.1.3 Ubiquitin

Ub* : Pseudo Wild Type (F45W)

< Amino acid sequence in 1 letter code >

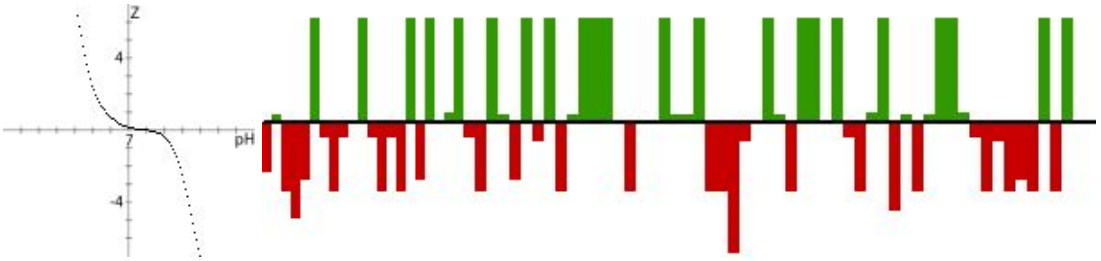
MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLI**W**AGKQLEDGRTL SDYNIQKESTLHLVLRLRGG

< Amino acid sequence in 3 letter code >

Met-Gln-Ile-Phe-Val-Lys-Thr-Leu-Thr-Gly-Lys-Thr-Ile-Thr-Leu-Glu-Val-Glu-Pro-Ser-Asp-Thr-Ile-Glu-Asn-Val-Lys-Ala-Lys-Ile-Gln-Asp-Lys-Glu-Gly-Ile-Pro-Pro-Asp-Gln-Gln-Arg-Leu-Ile-**Trp**-Ala-Gly-Lys-Gln-Leu-Glu-Asp-Gly-Arg-Thr-Leu-Ser-Asp-Tyr-Asn-Ile-Gln-Lys-Glu-Ser-Thr-Leu-His-Leu-Val-Leu-Arg-Leu-Arg-Gly-Gly

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
76	8603.9	6970	10410

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
0.1	7.7	0.3	43 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

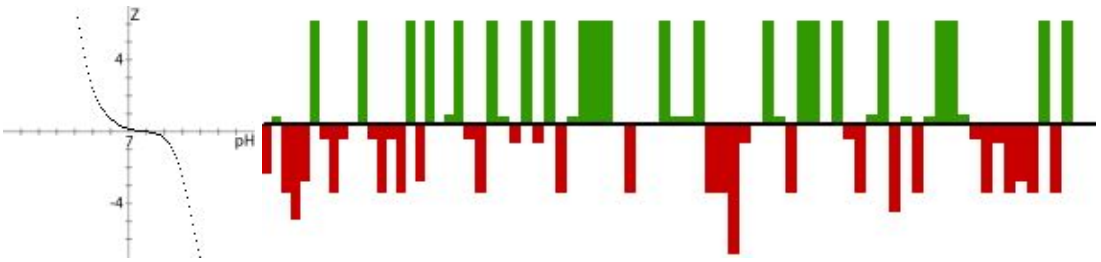
Ub* V26A (F45W V26A)

< Amino acid sequence in 1 letter code >

MQIFVKTLTGKTITLEVEPSDTIEN**A**KAKIQDKEGIPPDQQRLI**W**AGKQLEDGRTL SDYNIQKESTLHLVLRLRGG

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
76	8575.8	6970	10376

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
0.1	7.7	0.3	43 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

Ub* I61A (F45W I61A)

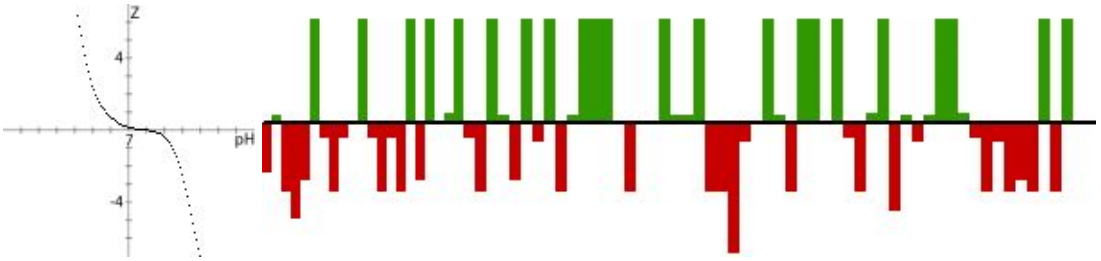
< Amino acid sequence in 1 letter code >

MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLI**W**AGKQLEDGRTL

SDYN**A**QKESTLHLVLRLRGG

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
76	8561.8	6970	10359

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
0.1	7.7	0.3	43 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

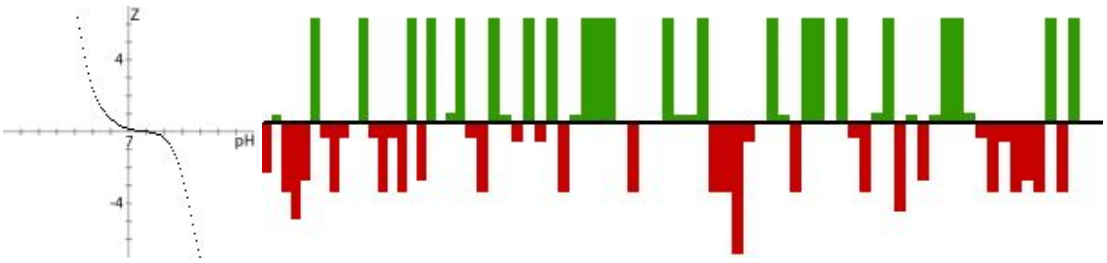
Ub* V26A I61V (F45W V26A I61V)

< Amino acid sequence in 1 letter code >

MQIFVKTLTGKTITLEVEPSDTIEN**A**KAKIQDKEGIPPDQQRLI**W**AGKQLEDGRTL SDYN**V**QKESTLHLVLRLRGG

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
76	8561.8	6970	10359

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
0.1	7.7	0.3	43 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

B.1.4 U1A

< Amino acid sequence in 1 letter code >

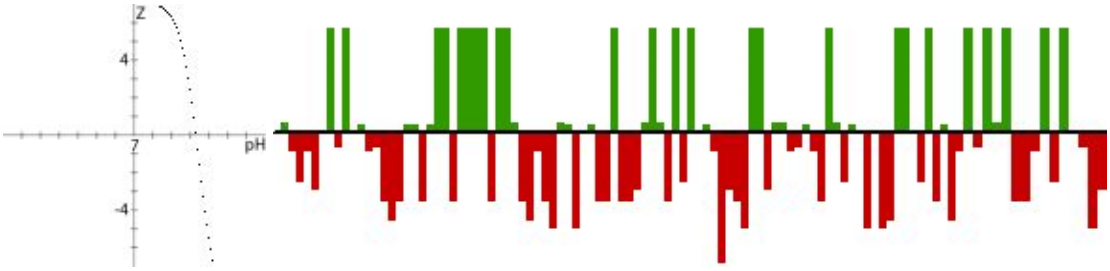
GSHMAVPETRPNHTIYINNLNEKIKKDELKKSLYAIFSQFGQILDILVSRSLKMRGQ
AWVIFKEVSSATNALRSMQGFPFYDKPMRIQYAKTDSDIIAKMKGTFV

< Amino acid sequence in 3 letter code >

Gly-Ser-His-Met-Ala-Val-Pro-Glu-Thr-Arg-Pro-Asn-His-Thr-Ile-Tyr-Ile-Asn-Asn-Leu-Asn-Glu-Lys-Ile-Lys-Lys-Asp-Glu-Leu-Lys-Lys-Ser-Leu-Tyr-Ala-Ile-Phe-Ser-Gln-Phe-Gly-Gln-Ile-Leu-Asp-Ile-Leu-Val-Ser-Arg-Ser-Leu-Lys-Met-Arg-Gly-Gln-Ala-Trp-Val-Ile-Phe-Lys-Glu-Val-Ser-Ser-Ala-Thr-Asn-Ala-Leu-Arg-Ser-Met-Gln-Gly-Phe-Pro-Phe-Tyr-Asp-Lys-Pro-Met-Arg-Ile-Gln-Tyr-Ala-Lys-Thr-Asp-Ser-Asp-Ile-Ile-Ala-Lys-Met-Lys-Gly-Thr-Phe-Val

Residues	Molecular Weight (g/mol)	Extinction Coefficient ($cm^{-1}M^{-1}$)	Approximate Volume ($Å^3$)
105	12044	10810	14573

Net Charges at pH 7.0	Iso-electric Point (pI)	Average Hydrophillicity	Ratio hydrophilic residues / total number of residues
7.2	10.2	0	42 %



Net Charge (left) and Hydrophilicity (right), from Innovagen peptide property calculator

B.2 How to grow proteins from plasmid DNA

B.2.1 The 1st day: Transformation

2.1.1 Fill a Styrofoam ice bucket with ice.

2.1.2 Take one small eppendorf containing an adequate *E. coli* cell (such as Rosetta TM (DE3) pLysS cells (Novagen) for the pET-15b vector) from the –80 °C freezer. Also take an eppendorf containing plasmid DNA from the deep freezer.

2.1.3 Extract 1 µL of the plasmid DNA solution (~100 ng/µL) by a pipet, and drop it into the *E. coli* cell eppendorf directly. Place the eppendorf in the ice bucket for 5 min, and put the remaining plasmid DNA solution back into the original position in the freezer.

2.1.4 Keep the *E. coli* eppendorf in a 42 °C water bath for 30 seconds. (42 °C is an optimal temperature for the *E. coli* cell to open its membrane.)

2.1.5 Place the *E. coli* eppendorf in the ice bucket for 2 minutes.

2.1.6 Add 500 µL of LB solution (nutrition for the bacteria) into the *E. coli* eppendorf.

2.1.7 Keep it in a shaker at 37 °C, for 5 minutes

2.1.8 Bend the tip of a thin Pasteur tube by a Bunsen burner flame, and sterilize it by fire and ethanol solution. Allow the Pasteur pipet to cool for 10 seconds before proceeding to the next step.

2.1.9 Take the eppendorf from the shaker and pour all the *E. coli* solution into an agar plate which contains the appropriate antibiotic (usually ampicillin or kanamycin). Spread the solution uniformly by using a bended tip (step 2.1.8).

2.1.10 Set the agar plate upright and keep it at room temperature for about 15 minutes. This time will allow the agar to absorb the *E. coli* suspension.

2.1.11 Place the agar plate into the 37 °C incubator, with the cover-side down, and keep it for 15 to 18 hours (overnight).

B.2.2 The 2nd day: Growing the *E. coli* cells and inducing the target proteins by IPTG

2.2.1 Check whether the colony grew well in the agar plate. Typically the colonies are small (white) spots on the surface of the agar plate. If you don't have enough colonies, wait more. If you still don't get any colony, please restart from the beginning of the first day protocol.

2.2.2 Prepare an antibiotic solution to exclusively grow the target *E. coli* cells containing the plasmid DNA. *E. coli* not having the plasmid DNA and other germs will be destroyed by the antibiotic.

For lambda repressor and SH3, prepare a 1000x ampicillin solution by dissolving 0.1 grams ampicillin sodium salt, MW 371.39, per 1 mL of water. Eventually it will be diluted down to 0.1 g/L (=269.3 μM) in LB broth. For example, 11 L of LB broth solutions (=11 flasks), we need to prepare 1.1 g of ampiciline solution in 11 mL of

distilled water.

For U1A, we use a kanamycin (M.W. 484.5) instead. 1000X concentrated solution is to be prepared at the concentration of 70 mg/mL. Eventually it will be diluted in LB solution down to 70 mg/L (=144.5 μM).

2.2.3 Pick up 20 μL of 1000X antibiotic (ampiciline or kanamycin) solution, and add it into the falcon tube containing 20 mL of LB broth.

2.2.4 Touch one of the grown colonies on the agar plate with a sterile tip very softly, dip the tip into the above LB broth, and shake well.

2.2.5 Keep the 20 mL LB broth in the shaker at 37 °C, and let *E. coli* cells grow for about 5 hours. The solution is getting cloudy as the *E. coli* cells grow. Its population doubles every 20~30 minutes.

2.2.6 Make 11 L of LB broth solutions that provides nutrition to *E. coli* cells

- Per every 1 L of distilled water, prepare
  - 10 g of Tryptone
  - 5 g of Bacto-Yeast Extract
  - 10 g of NaCl
- Adjust to pH 7.0, with 5 M NaOH (typically 500 μL)

2.2.7 Split LB broth into 11 large flasks, which are used for supplying enough air to the bacteria.

2.2.8 Wrap the top of each flask with an aluminum foil, and autoclave all 11 flasks at 120 °C for 25 minutes for the sterilization. Select the “Liquid” option, the whole procedure will take about 1 hour to finish.

2.2.9 Add 1 mL each of 1000x antibiotic solution (step 2.2.2) into each flask.

2.2.10 Transfer 1 mL of pre-grown LB solution (step 2.2.5) into each 37 °C or cooler flask.

2.2.11 Keep all 11 flasks in the shaker at 37 °C, and let *E. coli* cells grow until the optical depth (absorption) reaches 0.6 at 600 nm (typically it takes 5 to 6 hours).

2.2.12 Induce the target proteins by adding IPTG (isopropyl-β-D-thiogalactopyranoside, M.W. 238).
- Prepare 1000X concentrated IPTG solution (0.238 g/mL) by adding 0.238 g/mL * 11 mL = 2.62 g of IPTG into 11 mL of distilled water for all 11 flasks. The final goal concentration in LB broth is 1 mM.
- Apply IPTG solution when the temperature of the flaks drops down to around 30 °C. We can keep all the flasks outside to cool down for 30 minutes.

2.2.13 Change the temperature setting of the shaker to the minimum, and induce the target proteins for more than 10 hours (keep overnight). Keep slightly open the cover of the shaker by putting some paper sheets (or a Styrofoam container) to the edge of the cover for making a gap.

B.2.3 The 3rd day: Harvesting cells (centrifugation and cell breaking)

2.3.1 Centrifuge cloudy LB broth solutions from 11 flasks for 10 minutes at 5000 rpm, in 6 centrifugation bottles. *E. coli* cell pellets will be deposited on the bottom of each bottle. Put the supernants back into the flasks, and add some bleach before pouring it down the drain. (It smells awful.)

2.3.2 Collect all the cell pellets into 50 mL falcon tubes using a spatula and keep them in the freezer at least for several hours. This process will weaken cell membranes for easy extraction of the target proteins from the cell.

2.3.3 Bleach the supernant solutions and throw away. Wash all the flasks ever used.

2.3.4 Pour ~150 mL of a buffer solution into the cell pellets (step 2.3.2), with continuously transferring it into a 250 mL Erlenmeyer flask. We use a 10 mM Imidazole equilibrium buffer for a purification process (see the protein purification part below).

For a Ni-Agarose His-tag binding column, prepare at least 500 mL of

- 50 mM PO_4 (Na_2HPO_4, MW 141.96) : 7.1 g / L
- 500 mM NaCl (MW 58.44) : 29.22 g / L
- 10 mM Imidazole (MW 68.08) : 0.68 g / L
- PH 8.0 by adding small amounts of HCl.

2.3.5 Add a tiny amount (a spoon tip) of DNase I to cut a DNA into many fragments and digest them.

2.3.6 Fill a Styrofoam ice bucket with ice, and keep all the samples in it.

2.3.7 Break *E. coli* cells to extract the target proteins by a French Press. Repeat twice at a pressure of 1500 Psi. Don't apply higher pressure than 1600 Psi, to avoid any damage to the French Press.

2.3.8 Collect the flow-through from the French Press and centrifuge it for 25 minutes at 10,000 rpm using the JA-17 Beckman rotor. The target proteins are dissolved in supernants. Other cell junks will be deposited on the bottom of the centrifugation tubes. Then we are ready for the purification of proteins dissolved in the supernants.

B.2.4 Protein purification: Ni-Agarose His-tag binding column for lambda repressor and *fyn*-SH3

2.4.1 Regenerate the Ni-Agarose His-tag binding column (resin volume: ~20 mL) thoroughly per every five time usage. Refer to the instructions from the manufacturer.

2.4.2 Load about 40 mL (two times the volume) of 10 mM Imidazole buffer (step 2.3.4) into the column to pre-equilibrate the column with 10 mM Imidazole solution. We can use a peristaltic pump to expedite the buffer loading. Run the peristaltic pump at the speed of 150~200 mL/hour.

2.4.3 Load all supernants (step 2.3.8) into the Ni-Agarose column. Only the target proteins with a His-tag will be attached to the Ni-Agarose column.

2.4.4 Wash the column by flowing about 300~500 mL of 20 mM Imidazole buffer solution (PH 8.0) into cells until the absorption reaches 0.05 at 260 nm (It indicates the amount of DNA)

- 50 mM PO_4^{3-} (Na_2HPO_4, MW 141.96) : 7.1 g / L
- 500 mM NaCl (MW 58.44) : 29.22 g / L
- 20 mM Imidazole (MW 68.08) : 1.36 g / L
- Adjust to pH 8.0 by adding small amount of HCl

2.4.5 Elute the target proteins by loading small volumes of 250 mM Imidazole buffer solution (PH 8.0) into the column. Don't use the peristaltic pump for this step.

- 50 mM PO_4^{3-} (Na_2HPO_4, MW 141.96) : 7.1 g / L
- 500 mM NaCl (MW 58.44) : 29.22 g / L
- 250 mM Imidazole (MW 68.08) : 17.02 g / L
- Adjust to pH 8.0 by adding small amount of HCl.
- Check UV absorption at 250-340 nm frequently (Peak at 280 nm) to determine how much protein is being eluted.

B.2.5 Thrombin digestion for His-tag removal

2.5.1 Dialyze for at least 5 hours in the cold room at 4 °C to remove NaCl, Imidazole and everything other than protein. The following is a suitable buffer for dialysis.

- 50 mM PO_4^{3-} (Na_2HPO_4, MW 141.96) : 7.1 g / L (pH 7.0)

You can also use ultra-pure water if the protein is very stable.

2.5.2 Remove the His-tag by adding ~1 unit of thrombin per mg of the protein. Use restriction grade thrombin for digestion. Keep the digestion reaction at room temperature overnight (16 hours), or longer at 4°C. Check that digestion is complete by running an SDS-page (see Appendix B.3.2)

B.2.6 Protein purification: Cation exchange column (CM-52 column with cellulose matrix) for ubiquitin

2.6.1 Regenerate the column with one half column volume (~250 mL) of 0.5 M NaOH or 6 M Guanidine Hydrochloride solution

2.6.2 Equilibrate the column with 2 column volumes (1 L) of 50 mM Sodium-Acetate-Buffer at pH 5.0 until the pH of the solution passed through the column is pH 5.0 (check with pH paper)

2.6.3 Dialyze the supernatants (step 2.3.8) with 3 kDa MWCO dialysis tubing (Fisher Scientific) against 4 L of a 50 mM Sodium-Acetate-Buffer Solution at 4°C (cold room!):

- 50 mM Sodium-Acetate (MW 82.05) : 4.102 g / L
- 5 mM EDTA (MW 372.24) : 1.861 g / L
- Adjust to pH 5.0 by adding a small amount of 6M HCl

** Much Precipitate will occur due to aggregation, so only take the supernatant for further purification.

2.6.4 Centrifuge the above sample solution again in small tubes for 30 min at 10000 rpm, and then filter it through a 0.2 micron sterile filter.

2.6.5 Add the filtered Ubiquitin-Buffer-Solution to the CM-52 cellulose cation exchange column, then ubiquitin and DNA will be bound to the column.

2.6.6 Wash the column by adding about 500 mL of 50 mM Sodium-Acetate-Buffer (pH 5.0) until the absorption is 0.05 at 260 nm, which indicates a negligible amount of DNA. Other proteins and nucleic acids which are not of interest will be washed out by the 50 mM Sodium-Acetate-Buffer.

2.6.7 Elute Ubiquitin against a NaCl Salt-Gradient solution by adding 50 mM Sodium-Acetate-Buffer (pH 5.0) with 1 M NaCl and collect the fractions (50 mL each).

- 50 mM Sodium-Acetate (MW 82.05) : 4.102 g / L
- 5 mM EDTA (MW 372.24) : 1.861 g / L
- 1 M NaCl (MW 58.44) : 58.44 g / L
- pH 5.0 by adding a small amount of 6 M HCl
- Ubiquitin will be eluted from Fraction 3 to 10 (V_{total}~ 300mL)
- Check UV from 250-340 nm frequently (Peak at 280 nm)

B.2.7 Calculate the protein yield

2.7.1 Calculation of Concentration (ubiquitin, for example)

- Absorption value at the 280 nm peak: 1.294
- Baseline value at 310 nm: 0.447
- The difference is 1.294 – 0.447 = 0.847 = OD_{280}
- Absorption coefficient A_{280} [Trp, Tyr] = 6970 $M^{-1}cm^{-1}$ (depends on the protein and its mutations)
- Optical pathlength L = 1.0 cm

$OD_{280} = 0.847 = A_{280} \cdot L \cdot c = 6970\ M^{-1}cm^{-1} \cdot 1.0\ cm \cdot c$

Therefore, c = 0.847 / 6970 M^{-1} = 1.22 · 10^{-4} M

2.7.2 Calculate the Protein Yield

m = c (Concentration) · V (Volume) · M (Molecular Weight)

= 1.22 · 10^{-4} M · 0.020 L· 8472 g · mol^{-1} = 0.02067 g

The total yield of ubiquitin is 20.7 mg.

B.3 Protein purification and verification

B.3.1 Further purifications

* We need to use the SDS Gel electrophoresis or Mass Spectroscopy to check how the purity of the target proteins improves with each step.

3.1.1 Purify the proteins in 30 kDa MWCO Amicon at p = 350 kPa (50 PSI) in order to get rid of DNA, Thrombin and other junk in the sample solution. The target proteins (and particles having less than 30K M.W) will penetrate the size-selective membrane, so we need to collect the flow-through from the Amicon. Check the purity of the sample by SDS-page and mass spectroscopy.

3.1.2 Dialyze (using 3.5 kDa dialysis tubing) for at least 3 times for 5 hours each against 4L of ultra-pure water (or buffer) in the cold room at 4°C to remove NaCl, Nucleic Acids and everything other than the target proteins. Check the purity of the sample.

3.1.3 If necessary, run the sizing column in order to purify the protein further. First of all, concentrate the sample solution by using 3kDa MWCO (molecular weight cut-off) Amicon. Only a small volume (~ 5 mL) of protein sample can be purified with a sizing column at once.

3.1.4 Lyophilize the proteins: freeze the sample instantly by liquid nitrogen and bring it to the Lyophilizer on the 4th floor in Lu group. This will take approximately 3 days at the vacuum pressure of less than 1 Torr.

B.3.2 SDS gel electrophoresis

3.2.1 Attach the yellow rubber tube to the U-type glass, to prepare the gel frame

3.2.2 Separation Gel:
1.5 mL Acrylamide
1.5 mL Gel buffer
1.5 mL Glycerol
80 μL APS (10%): Ammonium Peroxydisulfate
10 μL TEMED (It makes the gel solution polymerize fast! Be careful…)

3.2.3 Loading Gel:
0.3 mL Acrylamide
0.75 mL Gel buffer
2.0 mL Water
80 μL APS (10%): Ammonium Peroxydisulfate
10 μL TEMED (It makes the gel solution polymerize fast! Be careful…)

3.2.4 Prepare
- 10 μL Load Buffer + 10 μL Sample
- 10 μL Load Buffer + 10 μL Reference Ladder (Kaleidoscope Polypeptide Standards)
→ Centrifuge solution to make uniform

3.2.5 Denature (unfold) proteins by heating up to 70°C for 10 minutes

3.2.6 Prepare buffer solution

- 5X Cathode buffer (upper) 20 mL + 80 mL water
- 10X Anode buffer (lower) 30 mL + 270 mL water

3.2.7 Apply sample and reference ladder into gel chambers

3.2.8 Apply 120V for about 3~4 hours or until the blue dye is 1/3 to 1/4 from the bottom of the gel.

3.2.9 Wash and Stain Gel with Simply Blue stain for 1 hour

3.2.10 Wash the Gel (rinse with water) and keep it in distilled water overnight to remove the background and intensify the band

B.3.3 Sizing column

3.3.1 Prepare about 3 mL of protein sample (maximum 5 mL), glass tubes and the frame.

3.3.2 Prepare 1 L of Buffer Solution

- 50 mM PO_4^{3-} (Na_2HPO_4, MW 141.96) : 7.098 g / L
- 300 mM NaCl (MW 58.44) : 17.532 g / L
- pH 7.0

3.3.3 Filter the buffer solution using a 0.22 μm filter

3.3.4 Equilibrate the sizing column with about 100mL of Buffer solution by using a Peristaltic pump at the speed of 150 mL/hour. Take care to ensure no air bubbles enter into the column.

3.3.5 Load sample solution by running the peristaltic pump from your sample tube.

3.3.6 Collect 95 drops per tube with the fraction collector (It will take 3 hours)
** lambda-repressor protein can be found between 50~60 tubes generally.

3.3.7 Wash the column with at least 200 mL of buffer (Step 3.3.2) for long term storage.

B.4 How to amplify a plasmid DNA

B.4.1 The 1st day: Transformation

4.1.1 Fill a Styrofoam ice bucket with ice.

4.1.2 Take one small eppendorf containing an adequate *E. coli* cell (such as BL21 and Rosetta TM (DE3) pLysS cells (Novagen) for the pET-15b vector) from the –80 °C freezer. Also take an eppendorf containing plasmid DNA from the deep freezer.

4.1.3 Extract 1 μL of the plasmid DNA solution (~100 ng/μL) by a pipet, and drop it into the *E. coli* cell eppendorf directly. Place the eppendorf in the ice bucket for 5 min, and put the remaining plasmid DNA solution back into the original position in the freezer.

4.1.4 Keep the *E. coli* eppendorf in a 42 °C water bath for 30 seconds. (42 °C is an optimal temperature for the *E. coli* cell to open its membrane.)

4.1.5 Place the *E. coli* eppendorf in the ice bucket for 2 minutes.

4.1.6 Add 500 μL of LB solution (nutrition for the bacteria) into the *E. coli* eppendorf.

4.1.7 Keep it in a shaker at 37 °C, for 5 minutes

4.1.8 Bend the tip of a thin Pasteur tube by a Bunsen burner flame, and sterilize it by

fire and ethanol solution. Allow the Pasteur pipet to cool for 10 seconds before proceeding to the next step.

4.1.9 Take the eppendorf from the shaker and pour all the *E. coli* solution into an agar plate which contains the appropriate antibiotic (usually ampicillin or kanamycin). Spread the solution uniformly by using a bended tip (step 4.1.8).

4.1.10 Set the agar plate upright and keep it at room temperature for about 15 minutes. This time will allow the agar to absorb the *E. coli* suspension.

4.1.11 Place the agar plate into the 37 °C incubator, with the cover-side down, and keep it for 15 to 18 hours (overnight).

B.4.2 The 2nd day: Growing the *E. coli* cells

4.2.1 Check whether the colony grew well in the agar plate. Typically the colonies are small (white) spots on the surface of the agar plate. If you don't have enough colonies, wait more. If you still don't get any colony, please restart from the beginning of the first day protocol.

4.2.2 Prepare an antibiotic solution to exclusively grow the target *E. coli* cells containing the plasmid DNA. *E. coli* not having the plasmid DNA and other germs will be destroyed by the antibiotic.

For lambda repressor and SH3, prepare a 1000x ampicillin solution by dissolving 0.1 grams ampicillin sodium salt, MW 371.39, per 1 mL of water. Eventually it will be diluted down to 0.1 g/L (=269.3 μM) in LB broth. For example, 11 L of LB broth solutions (=11 flasks), we need to prepare 1.1 g of ampiciline solution in 11 mL of distilled water.

For U1A, we use a kanamycin (M.W. 484.5) instead. 1000X concentrated solution is to be prepared at the concentration of 70 mg/mL. Eventually it will be diluted in LB solution down to 70 mg/L (=144.5 μM).

4.2.3 Pick up 20 μL of 1000X antibiotic (ampiciline or kanamycin) solution, and add it into the falcon tube containing 20 mL of LB broth.

4.2.4 Touch one of the grown colonies on the agar plate with a sterile tip very softly, dip the tip into the above LB broth, and shake well.

4.2.5 Keep the 20 mL LB broth in the shaker at 37 °C, and let *E. coli* cells grow for a day. The solution is getting cloudy as the *E. coli* cells grow. Its population doubles every 20~30 minutes.

B.4.3 The 3rd day: Extraction of plasmid DNA and purification

4.3.1 Centrifuge 20 mL of LB broth for 30 minutes at maximum speed in Clinical Centrifuge machine (in 3rd floor, Lu group), or centrifuge in A229 CLSL.

4.3.2 Dispose supernants and keep the cultures deposited on the bottom of the tube in the freezer for a while. This procedure will weaken the cell membranes to expedite DNA extraction.

4.3.3 Follow an instruction from a QIAGEN kit.

4.3.4 Check the sequence of plasmid DNA at the biotech center in the university.

B.5 Site-directed mutagenesis

B.5.1 The 1st day: Design and order a primer

5.1.1 Design your Primer sets from the following website,

PrimerX - http://bioinformatics.org/primerx/

- Normally it consists of 25 ~ 45 bases pairs.
- Melting Temperature should be higher than 78 °C
- The primer should have GC content of about 40%.
- The primer should terminate in one or more C or G bases.
- Design a pair of primers which flank your gene and anneal to opposite strands.

5.1.2 Order the primer at Biotech Center, ask Custom Oligonucleotide Synthesis with settings of 40 nmol and OPC Purified.

B.5.2 The 2nd day: Thermal cycling

5.2.1 Prepare ice in a bucket and keep all elements of a Stratagene Quik-Change kit in the ice bucket.

5.2.2 Measure UV absorption and estimate the concentration of primer and target plasmid DNA (1 OD_{260} ~ 50 ng / μL).

5.2.3 Prepare 125 ng / μL of Primer sets and 10 ng / μL of target-Plasmid DNA

5.2.4 Prepare 4 small autoclaved eppendorfs, as the following table.

Eppendorf #	#1	#2	#3	#4
Target Plasmid DNA	1 µL	2 µL	3 µL	4 µL
10X Reaction Buffer	5 µL	5 µL	5 µL	5 µL
Primer 5' → 3'	1 µL	1 µL	1 µL	1 µL
Primer 3' → 5'	1 µL	1 µL	1 µL	1 µL
dNTP	1 µL	1 µL	1 µL	1 µL
double-distilled water (dd H_2O)	41 µL	40 µL	39 µL	38 µL
PfuTurbo DNA polymerase	1 µL	1 µL	1 µL	1 µL
Mineral Oil	30 µL	30 µL	30 µL	30 µL

5.2.5 Centrifuge all the eppendorfs at the speed 4, for 5 minutes.

5.2.6 Start a Thermal Cycling (use Program #4 on the memory of the instrument)

Block #	#1	#2	#3	#4
Temperature	95 °C	55 °C	68 °C	20 °C
Minutes	0:30	1:00	7:00	-

Point mutations – 12 cycles
Single Amino Acid Changes – 16 cycles
Multiple Amino Acid Deletions or insertions – 18 cycles

5.2.7 Add 1 µL of Dpn I using small pipets for PCR ONLY. Avoid touching oil layers on the top. Put Dpn I below the oil layer.

5.2.8 Centrifuge the samples for 1 minute.

5.2.9 Incubate at 37 °C, for an hour to digest the parental (non-mutated) supercoiled dsDNA.

5.2.10 Remove oil layers before transformation. Use a small pipet.

B.5.3 The 3rd day: Transformation

5.3.1 Fill a Styrofoam ice bucket with ice.

5.3.2 Get a XL1-Blue supercompetent cell from –80 °C deep freezer, and gently thaw in ice.

5.3.3 Pipet 50 μL of the cell to four pieces of 14 mL sterile eppendorfs.

5.3.4 Extract 1 μL of each Dpn I-treated DNA solution (~100 ng/μL) by a pipet, and drop it into each XL1-Blue supercompetent cell directly. Place all the eppendorfs in the ice bucket for 5 minutes.

5.3.5 Keep all the eppendorfs in a 42 °C water bath for 45 seconds. (42 °C is an optimal temperature for the *E. coli* cell to open its membrane.)

5.3.6 Place all the eppendorfs in the ice bucket for 2 minutes.

5.3.7 Add 500 μL of LB solution (nutrition for the bacteria) into each eppendorf.

5.3.8 Keep it in a shaker at 37 °C, for 1 hour at least. (at ~200 rpm)

5.3.9 Bend the tip of a thin Pasteur tube by a Bunsen burner flame, and sterilize it by fire and ethanol solution. Allow the Pasteur pipet to cool for 10 seconds before proceeding to the next step.

5.3.10 Take all the eppendorfs from the shaker and pour each *E. coli* solutions into each agar plate which contains the appropriate antibiotic (usually ampicillin or kanamycin). Spread the solution uniformly by using the bended tip (step 5.3.9)

5.3.11 Set all the agar plates upright and keep them at room temperature for about 15 minutes. This time will allow the agar to absorb the *E. coli* suspensions.

5.3.12 Place all the agar plates into the 37 °C incubator, with the cover-side down, and keep them for 15 to 18 hours (overnight).

B.5.4 The 4th day: Amplify the plasmid DNA

- Follow the instructions in [B.4.2] and [B.4.3].

Appendix C Program codes in C language

C.1 Simulation-based fitting of protein-protein interaction potentials to SAXS experiments

C.1.1 Main function (sax_agg.c)

The main function reads a configuration file (.INI file), determines the execution mode, and calls an appropriate function.

```
//=========================================================================
// Main function
//=========================================================================
int main(int argc, char *argv[]) {
        time_t s1, s2;
        int Levenberg_Marquardt, i;
        struct tm *newTime;
        time_t szClock;

        // Get UNIX-style time and display as number and string.
        time(&szClock);
        newTime = localtime(&szClock);
        printf("%s", asctime(newTime));

        g_argc = argc - 1;    // Number of TXT files

        if (g_argc >= 1) {
                s1 = time (NULL);
                Levenberg_Marquardt = ReadINIparameters(argv[1]);
                g_Concentration *= 1.0e-6;
                g_CubeSize = pow(n_particles/(1000.0*g_Concentration*AvogadroConstant()), 1.0/3.0);
                SetgRunning(1);

                g_Concs = (double*)calloc(g_argc, sizeof(double));
                g_CubeSizeS = (double*)calloc(g_argc, sizeof(double));
                g_InputFiles = (char **)calloc(g_argc, sizeof(char *));
                g_OutputFiles = (char **)calloc(g_argc, sizeof(char *));

                for (i=0; i<g_argc; i++) {
                        g_InputFiles[i] = (char *)calloc(256, sizeof(char));
                        g_OutputFiles[i] = (char *)calloc(256, sizeof(char));

                        ReadMultipleINIparameters(argv[i+1], &g_Concs[i], &g_OutputFiles[i],
                        &g_InputFiles[i]);
                        g_Concs[i] *= 1.0e-6;
                        g_CubeSizeS[i] = pow(n_particles/(1000.0*g_Concs[i]*AvogadroConstant()), 1.0/3.0);
                }
                if (g_argc >= 2) {    // Multiple File Input
                /*if ( Levenberg_Marquardt != 0 && Levenberg_Marquardt != 50 && Levenberg_Marquardt != 98
```

```
                && Levenberg_Marquardt != 99)
                Levenberg_Marquardt++;*/
        }

        // Determine the execution mode
        switch ( Levenberg_Marquardt ) {
                // No Optimization, just do Metropolis simulation at once.
                case 0 :    Metropolis(1);
                            break;

                // Metropolis simulation with Levenberg-Marquardt Optimization
                case 1 :    Levenberg_Marquardt_Metropolis(1);
                            break;
                // -1 : with Rg FIXED !!
                case 10 :   Levenberg_Marquardt_Metropolis(-1);
                            break;
                // -2 : with D0 FIXED !! (D0 = 2*sqrt(5/3)Rg * PW)) !!
                case 100 :          Levenberg_Marquardt_Metropolis(-2);
                            break;
                // -3 : with D0 FIXED !! (D0 = 2*sqrt(5/3)Rg * PW)) !! && PR fixed!!
                case 110 :          Levenberg_Marquardt_Metropolis(-3);
                            break;

        // Metropolis simluation with Levenberg-Marquardt Optimization, for multiple concentrations
                case 1 :    Multiple_Levenberg_Marquardt_Metropolis(1);
                            break;
                // -1 : with Rg FIXED !!
                case 10 :   Multiple_Levenberg_Marquardt_Metropolis(-1);
                            break;
                // -2 : with D0 FIXED !! (D0 = 2*sqrt(5/3)Rg * PW)) !!
                case 100 :Multiple_Levenberg_Marquardt_Metropolis(-2);
                            break;
                // -3 : with D0 FIXED !! (D0 = 2*sqrt(5/3)Rg * PW)) !! && PR fixed!!
                case 110 :Multiple_Levenberg_Marquardt_Metropolis(-3);
                            break;

                // Winter's simulation
                case 50 :   Winter(1);   // No Optimization, just do Winter's simulation at once.
                            break;
                case 500 :Multiple_LM_Winter(1);       // for Multiple concentrations
                            break;
                // -2 : with D0 FIXED !! (D0 = 2*sqrt(5/3)Rg * PW) !!
                case 510 :Multiple_LM_Winter(-2);
                            break;
                // -3 : with D0 FIXED !! (D0 = 2*sqrt(5/3)Rg * PW) && PR fixed!!
                case 520 :Multiple_LM_Winter(-3);
                            break;

                // Molecular Dynamics simulation in a given potential model
                case 98 :   MDsimulation(0);                        // Verlet's Method
                            break;
                case 99 :   MDsimulation(1);                        // Beeman's Method
                            break;

                default :   break;
        }
        s2 = time (NULL);
        printf("\nTotal Execution Time : %g hours (=%g minutes)\n", (s2-s1)/3600.0, (s2-s1)/60.0);
}
else
        printf("Please Input INI file name!\n");
```

```
	for (i=0; i<g_argc; i++) {
		Destroy(g_InputFiles[i]);
		Destroy(g_OutputFiles[i]);
	}
	Destroy(g_Concs);
	Destroy(g_CubeSizeS);
	Destroy(g_InputFiles);
	Destroy(g_OutputFiles);
	return 0;
}
```

C.1.2 Simple Metropolis Monte Carlo simulation (saxs_agg.c)

This simple Metropolis function simulates the protein aggregation and calculates an expected SAXS profile for a single concentration by Metropolis Monte Carlo method, under the given potential. The optimization of protein-protein interaction potential (Levenberg-Marquardt algorithm) was not applied to this module.

```
//=============================================================================
// Metropolis Monte Carlo Simulation (No LM Optimization), just for a single concentration
//=============================================================================
int Metropolis(int panel) {
	MDdata *md;
	int count=0, count_Guinier=1, count_Temperature=0, i, count_Metropolis=1, Metropolis_repeat;
	double time=0.0, variable_size_factor, chisq=0.0;
	FILE *fp;
	int result, rejection=0, n_array;
	double RMS_Error, RSE, fittedArray[N_DATA_SET], coefficientArray[1] = {0.0};

	md = (MDdata*) calloc(n_particles, sizeof(MDdata));
	initialize(panel, md);
	g_Potential = (double*)malloc(sizeof(double)*g_PotentialBinSize);
	PreCalculate_Potential(g_Potential, g_PotentialDepth, g_PotentialRange, g_PotentialWall,
		g_EFFECTIVE_FORCE_RANGE_2, &g_coeff, g_Repulsive_Potential_Depth,
		g_Repulsive_Potential_Range);

	Metropolis_repeat = n_particles * g_MP_Repeat;
	variable_size_factor = Random_Distribution(panel, md, g_CubeSize);

	while ( (g_Running == 1) && (count_Guinier <= g_max_repeat) ) {
		for (count_Metropolis=1; count_Metropolis<=Metropolis_repeat; count_Metropolis++) {
			result = Metropolis_Sampling(panel, md, variable_size_factor, (int)NR_Random(0.0,
			n_particles, &g_Seed_MC2));
		}

		SAXS_Guinier(panel, &count_Guinier, md, variable_size_factor);
		variable_size_factor = Random_Distribution(panel, md, g_CubeSize);
		count_Guinier++;
	}

	#ifdef N_DATA_SET
		n_array = N_DATA_SET;
```

```
	#else
		n_array = (int)((g_qmax-g_qmin)/g_dq + 1.0);
	#endif

	// Offset Optimization for the least chi-square value
	RMS_Error = Offset_Optimization(&RSE, coefficientArray, fittedArray);
	fp=fopen(g_OutputFile, "w");
	fprintf(fp, "%d\t%.10lf\t%.10lf\t%.10lf\n", count_Guinier-1, RSE, RMS_Error, coefficientArray[0]);
	for (i=0; i<n_array; i++) {
		fprintf(fp, "%.10f\t%.10f\n", g_qq[i], (g_log_Iaverage[i] + coefficientArray[0]));
	}
	fclose(fp);
	finalize(panel, md);

	return 0;
}


//=============================================================================
// Randomly relocate the proteins by Random Number Generators
//=============================================================================
double Random_Distribution(int panel, MDdata *md, double iCubeSize) {
	int i=0, flag;
	double CubeSize, variable_size_factor = 1.0;

	variable_size_factor = NR_Random(pow(0.5, 1.0/3.0), pow(1.5, 1.0/3.0), &g_Seed_VariableSize);
	CubeSize = iCubeSize * variable_size_factor;

	while ( i < n_particles ) {
		#ifdef CUBIC_UNIT
			md[i].x = NR_Random(0.0, 1.0, &g_Seed_xyz) * CubeSize;
			md[i].y = NR_Random(0.0, 1.0, &g_Seed_xyz) * CubeSize;
			md[i].z = NR_Random(0.0, 1.0, &g_Seed_xyz) * CubeSize;
		#endif

		if (g_isHardSphere) {
			int j;
			double rr, PotentialWall_2 = g_PotentialWall*g_PotentialWall;
			r_vector dr;

			for (j=0; j<i; j++) {
				find_rr(i, j, md, variable_size_factor, &rr, &dr);
				if (rr < PotentialWall_2) {
					flag = 0;
					break;
				}
			}
		}

		if (flag == 1) {
		i++;
		}
	}
	return variable_size_factor;
}


//=============================================================================
// Distribute proteins according to a potential model, for a single Concentration (without Levenberg-
marquardt)
//=============================================================================
int Metropolis_Sampling(int panel, MDdata *md, double variable_size_factor, int index) {
```

```
	double energy1=0.0, energy2=0.0, delta_U, tempx, tempy, tempz;
	double CubeSize = g_CubeSize*variable_size_factor, u=0.0;
	int j;

	double rr, reciprocal_CubeSize = 1.0/CubeSize;
	r_vector dr;

	tempx = NR_Random(-g_step, g_step, &g_Seed_xyz);
	tempy = NR_Random(-g_step, g_step, &g_Seed_xyz);
	tempz = NR_Random(-g_step, g_step, &g_Seed_xyz);

	for (j=0; j<n_particles; j++) {
		if (j != index) {
			find_rr(index, j, md, variable_size_factor, &rr, &dr);

			if (rr < g_EFFECTIVE_FORCE_RANGE_2)   // using nearest separation rule
				energy1 += g_Potential[(int)(rr * g_coeff)];
		}
	}

	md[index].x_old = md[index].x;
	md[index].y_old = md[index].y;
	md[index].z_old = md[index].z;

	md[index].x += tempx;
	md[index].y += tempy;
	md[index].z += tempz;

	// Periodic Boundary Condition
	if (md[index].x >= CubeSize || md[index].x < 0.0 )
		md[index].x -= CubeSize * FLOOR(md[index].x * reciprocal_CubeSize);
	if (md[index].y >= CubeSize || md[index].y < 0.0)
		md[index].y -= CubeSize * FLOOR(md[index].y * reciprocal_CubeSize);
	if (md[index].z >= CubeSize || md[index].z < 0.0)
		md[index].z -= CubeSize * FLOOR(md[index].z * reciprocal_CubeSize);

	for (j=0; j<n_particles; j++) {
		if (j != index) {
			find_rr(index, j, md, variable_size_factor, &rr, &dr);

			if (rr < g_EFFECTIVE_FORCE_RANGE_2)   // using nearest separation rule
				energy2 += g_Potential[(int)(rr * g_coeff)];
		}
	}
	delta_U = energy2 - energy1;

	if (delta_U >= 0.0) {
		// Rollback; Rejected
		if ( NR_Random(0.0, 1.0, &g_Seed_MC) > exp( delta_U * g_reciprocal_of_KT ) ) {
		md[index].x = md[index].x_old;
			md[index].y = md[index].y_old;
			md[index].z = md[index].z_old;

			return -1;
		}
	}
	return 0;
}
```

C.1.3 SAXS calculation (saxs_agg.c)

This module calculates the expected (average) SAXS profiles per a given snapshot of the proteins.

```
//=============================================================================
// Calculate and average SAXS scattering profiles, in Guinier plot
//=============================================================================
int SAXS_Guinier(int panel, int *count_Guinier, MDdata *md, double variable_size_factor) {
	int i;

	for (i=0; i<N_DATA_SET; i++)
		g_SumI[i] += Calc_Scattering_Intensity(i, md, variable_size_factor, g_CubeSize);

	if (*count_Guinier - (int)(*count_Guinier/g_SAXS_MP_rate)*g_SAXS_MP_rate == 0) {
		FILE *fp;
		double RMS_Error, RSE, fittedArray[N_DATA_SET], coefficientArray[1] = {0.0};

		for (i=0; i<N_DATA_SET; i++) {
			g_log_Iaverage[i] = log( g_I[i] * g_SumI[i] / (double)(*count_Guinier) *
					g_volume_correction );
		}

		// Added by SJ Kim, 11/17/2007
		RMS_Error = Offset_Optimization(&RSE, coefficientArray, fittedArray);

		// Offset Optimization for the least chi-square value
		fp = fopen(g_OutputFile, "w");
		fprintf(fp, "%d\t%.10lf\t%.10lf\n", *count_Guinier, RSE, RMS_Error);
		for (i=0; i<N_DATA_SET; i++)
			fprintf(fp, "%.10f\t%.10f\n", g_qq[i], (g_log_Iaverage[i] + coefficientArray[0]));

		fclose(fp);
	}
	return 0;
}


//=============================================================================
// Calculate raw SAXS scattering profiles
//=============================================================================
double Calc_Scattering_Intensity(int index, MDdata *md, double variable_size_factor, double iCubeSize) {
	int i, count=0;
	fcomplex amplitude=complex(0.0, 0.0), intensity=complex(0.0, 0.0);
	double temp, psi=DegToRad(0.0), sin_psi, cos_psi, dpsi=DegToRad(45.0);

	double CubeSize, rr, Center;
	double dx, dy, dz;

	CubeSize = iCubeSize * variable_size_factor;
	Center = CubeSize*0.5;
	rr = Center * Center;

#ifdef PSI_AVERAGE
	while (psi < DegToRad(90.0)) {
#endif
		amplitude=complex(0.0, 0.0);
```

```
                    sin_psi = sin(psi);
                    cos_psi = cos(psi);
                    for (i=0; i<n_particles; i++) {
                            dx = md[i].x - Center;
                            dy = md[i].y - Center;
                            dz = md[i].z - Center;

                            if ((dx*dx + dy*dy + dz*dz) > rr)
                                    continue;

                            temp = -( - g_q_sin_theta[index] * md[i].x +
                                    g_q_cos_theta[index] * sin_psi * md[i].y +
                                    g_q_cos_theta[index] * cos_psi * md[i].z );
                            amplitude.r += cos(temp);
                            amplitude.i += sin(temp);
                    }
                    intensity = Cadd(intensity, Cmul(amplitude, Conjg(amplitude)));
                    count++;
#ifdef PSI_AVERAGE
                    psi += dpsi;
            }
#endif
            return (intensity.r/count);
}
```

C.1.4 Molecular Dynamics simulation (saxs_agg.c)

This simple Molecular Dynamics function simulates the protein aggregation and calculates an expected SAXS profile for a single concentration under the given potential. The optimization of protein-protein interaction potential (Levenberg-Marquardt algorithm) was not applied to this simple module. One can decide between Beeman's method and Verlet's method for time integration. Verlet's method normally guarantees faster calculation, but with less accuracy. For long time integrations, Beeman's method is recommended although it's slow. The automated Verlet's list is used for a faster calculation, and it is regularly updated. This module shares the same SAXS calculation module with Metropolis Monte Carlo function.

```
//=============================================================================
// Molecular Dynamics Simulation, in a given potental model, just for a single concentration
//=============================================================================
int MDsimulation(int panel) {
            MDdata *md;
            int count=0, count_Guinier=1, count_Temperature=0, i, count_MD=0;
            double variable_size_factor, chisq=0.0, temperature_sum=0.0;
            FILE *fp;
            int result=0, rejection=0, n_array=N_DATA_SET;
            double RMS_Error, RSE, fittedArray[N_DATA_SET], coefficientArray[1] = {0.0}, mass=MASS;
```

```
	md = (MDdata*) calloc(n_particles, sizeof(MDdata));
	g_Potential = (double*)malloc(sizeof(double)*g_PotentialBinSize);
	g_Force = (double*)malloc(sizeof(double)*g_PotentialBinSize);
	g_r = (double*)malloc(sizeof(double)*g_PotentialBinSize);
	g_Force_over_r = (double*)malloc(sizeof(double)*g_PotentialBinSize);
	g_acceleration_over_r = (double*)malloc(sizeof(double)*g_PotentialBinSize);

	initialize(panel, md);
	PreCalculate_Potential(g_Potential, g_PotentialDepth, g_PotentialRange, g_PotentialWall,
		g_EFFECTIVE_FORCE_RANGE_2, &g_coeff, g_Repulsive_Potential_Depth,
		g_Repulsive_Potential_Range);
	PreCalculate_Force(g_Potential, g_Force, g_r, g_Force_over_r, g_acceleration_over_r);

	variable_size_factor = Random_Distribution_MD(panel, md);

	while ( (g_Running == 1) && (count_Guinier <= g_max_repeat) ) {
		count_MD = 0;

		if (panel) {
			// Repeat many times...
			while ( (g_Running == 1) && (count_MD < g_MD_Repeat) ) {
				// Beeman's Method for Time Integration
				result = Beeman(md, variable_size_factor);
				count_MD++;
				if ( (int)(count_MD * 0.04) * 25 == count_MD )
					UpdateVerletList(md, variable_size_factor);
			}
		} else {
			// Repeat many times...
			while ( (g_Running == 1) && (count_MD < g_MD_Repeat) ) {
				// Verlet's Method for Time Integration
				result = Verlet(md, variable_size_factor);
				count_MD++;
				if ( (int)(count_MD * 0.04) * 25 == count_MD )
					UpdateVerletList(md, variable_size_factor);
			}
		}

		SAXS_Guinier(panel, &count_Guinier, md, variable_size_factor);
		variable_size_factor = Random_Distribution_MD(panel, md);
		count_Guinier++;
	}

	// Offset Optimization for the least chi-square value
	RMS_Error = Offset_Optimization(&RSE, coefficientArray, fittedArray);
	fp = fopen(g_OutputFile, "w");
	fprintf(fp, "%d\t%.10lf\t%.10lf\n", count_Guinier-1, RSE, RMS_Error);
	for (i=0; i<n_array; i++) {
		fprintf(fp, "%.10f\t%.10f\n", g_qq[i], (g_log_Iaverage[i] + coefficientArray[0]));
	}
	fclose(fp);
	finalize(panel, md);

	return 0;
}


//===========================================================================
// randomly relocate proteins by Random Number Generators for a Single Concentration in Molecular Dynamics
//===========================================================================
double Random_Distribution_MD(int panel, MDdata *md) {
```

```
int i=0, j, flag;
double variable_size_factor = 1.0;
double rr, CubeSize, spacing_allowance;
r_vector dr, a;
double v_average;                // max initial speed displacement

variable_size_factor = NR_Random(pow(0.5, 1.0/3.0), pow(1.5, 1.0/3.0), &g_Seed_VariableSize);

CubeSize = g_CubeSize * variable_size_factor;
v_average = sqrt( BOLTZMANN_COEFF * g_Temperature / MASS );
spacing_allowance = g_PotentialWall * g_PotentialWall;

#ifdef SPHERICAL_UNIT
        radius = CubeSize * g_radiusCoeff;
        rr = radius*radius;
        Center = CubeSize * 0.5;
#endif

while ( i < n_particles ) {
        flag = 1;

        #ifdef CUBIC_UNIT
                md[i].x = NR_Random(0.0, 1.0, &g_Seed_xyz) * CubeSize;
                md[i].y = NR_Random(0.0, 1.0, &g_Seed_xyz) * CubeSize;
                md[i].z = NR_Random(0.0, 1.0, &g_Seed_xyz) * CubeSize;
        #endif

        // Avoid Overlapping of proteins
        for (j=0; j<i; j++) {
                find_rr(i, j, md, variable_size_factor, &rr, &dr);

                if (rr <= 1.1*spacing_allowance) {
                        flag = 0;  // Reject it in the case of overlapping with any other proteins.
                        break;
                }
        }

        if (flag == 1) {        // Accept it
                // Velocity (Random Gaussian Distribution)
                md[i].vx = v_average * gasdev(&g_Seed_speed);
                md[i].vy = v_average * gasdev(&g_Seed_speed);
                md[i].vz = v_average * gasdev(&g_Seed_speed);

                md[i].x_old = md[i].x - md[i].vx * g_dT;          // the previous one..
                md[i].y_old = md[i].y - md[i].vy * g_dT;          // the previous one..
                md[i].z_old = md[i].z - md[i].vz * g_dT;          // the previous one..

                i++;
        }
}

UpdateVerletList(md, variable_size_factor);

if (panel) {
        for (i=0; i<n_particles; i++) {
                acceleration(i, md, &a, variable_size_factor, 1);   // current time
                md[i].ax = a.x;
                md[i].ay = a.y;
                md[i].az = a.z;
                md[i].x_old += 0.5 * (a.x) * g_dTdT;
                md[i].y_old += 0.5 * (a.y) * g_dTdT;
                md[i].z_old += 0.5 * (a.z) * g_dTdT;
```

```
                md[i].vx_old = md[i].vx - a.x * g_dT;
                md[i].vy_old = md[i].vy - a.y * g_dT;
                md[i].vz_old = md[i].vz - a.z * g_dT;
        }
        for (i=0; i<n_particles; i++) {
                acceleration(i, md, &a, variable_size_factor, 0);   // old time
                md[i].ax_old = a.x;
                md[i].ay_old = a.y;
                md[i].az_old = a.z;
        }
    }

    return variable_size_factor;
}


//============================================================================
// Update Verlist's list
//============================================================================
int UpdateVerletList(MDdata *md, double variable_size_factor) {
    int i, j;
    double rr;
    r_vector dr;

    for (i=0; i<n_particles; i++) {
        g_VerletList[i][0] = 0;
        /*for (j=0; j<n_particles; j++) {
            g_VerletList[i][j] = 0;
        }*/
    }

    for (i=0; i<n_particles; i++) {
        for (j=i+1; j<n_particles; j++) {
            find_rr(i, j, md, variable_size_factor, &rr, &dr);
            // Update Verlet Neightbor List Here
            if (rr < g_EFFECTIVE_FORCE_RANGE_2) {
                g_VerletList[i][ ++(g_VerletList[i][0]) ] = j;
                g_VerletList[j][ ++(g_VerletList[j][0]) ] = i;
            }
        }
    }
    return 0;
}


//============================================================================
// Verlet's Method for Molecular Dynamics Simulation
//============================================================================
int Verlet(MDdata *md, double variable_size_factor) {
    int i;
    r_vector f;
    double CubeSize = g_CubeSize * variable_size_factor, displacement, reciprocal_CubeSize = 1.0/CubeSize;
    double mass=MASS, *fx, *fy, *fz;

    fx = (double*)malloc(sizeof(double)*n_particles);
    fy = (double*)malloc(sizeof(double)*n_particles);
    fz = (double*)malloc(sizeof(double)*n_particles);

    // Calculate force on each particle
    for (i=0; i<n_particles; i++) {
        force(i, md, &f, variable_size_factor);
        fx[i] = f.x;        fy[i] = f.y;        fz[i] = f.z;
```

```
	}

	for (i=0; i<n_particles; i++) {
		// use Verlet method (g_coeff1 = (g_dT*g_dT) / mass);
		md[i].x_new = 2.0*md[i].x - md[i].x_old + fx[i] * g_coeff1;
		md[i].y_new = 2.0*md[i].y - md[i].y_old + fy[i] * g_coeff1;
		md[i].z_new = 2.0*md[i].z - md[i].z_old + fz[i] * g_coeff1;

		// keep tracks of velocities (g_coeff2 = 1.0 / (2.0*g_dT))
		md[i].vx = (md[i].x_new - md[i].x_old) * g_coeff2;
		md[i].vy = (md[i].y_new - md[i].y_old) * g_coeff2;
		md[i].vz = (md[i].z_new - md[i].z_old) * g_coeff2;

		// Periodic Boundary Condition
		if (md[i].x_new >= CubeSize || md[i].x_new < 0.0) {
			displacement = CubeSize * FLOOR(md[i].x_new * reciprocal_CubeSize);
			md[i].x     -= displacement;
			md[i].x_new -= displacement;
		}
		if (md[i].y_new >= CubeSize || md[i].y_new < 0.0) {
			displacement = CubeSize * FLOOR(md[i].y_new * reciprocal_CubeSize);
			md[i].y     -= displacement;
			md[i].y_new -= displacement;
		}
		if (md[i].z_new >= CubeSize || md[i].z_new < 0.0) {
			displacement = CubeSize * FLOOR(md[i].z_new * reciprocal_CubeSize);
			md[i].z     -= displacement;
			md[i].z_new -= displacement;
		}
	}

	// update current and old values
	for (i=0; i<n_particles; i++) {
		md[i].x_old = md[i].x;
		md[i].y_old = md[i].y;
		md[i].z_old = md[i].z;

		md[i].x = md[i].x_new;
		md[i].y = md[i].y_new;
		md[i].z = md[i].z_new;
	}

	free(fx);
	free(fy);
	free(fz);

	return 0;
}


//===========================================================================
// Beeman's Method for Molecular Dynamics Simulation
//===========================================================================
int Beeman(MDdata *md, double variable_size_factor) {
	int i, flag, count;
	r_vector a;
	double CubeSize = g_CubeSize * variable_size_factor, displacement, reciprocal_CubeSize = 1.0/CubeSize;
	double vx_prediction, vy_prediction, vz_prediction, vx_prediction_old, vy_prediction_old,
	vz_prediction_old;

	for (i=0; i<n_particles; i++) {
		md[i].x_new = md[i].x + md[i].vx*g_dT + c1*md[i].ax - c2*md[i].ax_old;
```

```
		md[i].y_new = md[i].y + md[i].vy*g_dT + c1*md[i].ay - c2*md[i].ay_old;
		md[i].z_new = md[i].z + md[i].vz*g_dT + c1*md[i].az - c2*md[i].az_old;

		// Periodic Boundary Condition
		if (md[i].x_new >= CubeSize || md[i].x_new < 0.0) {
			displacement = CubeSize * FLOOR(md[i].x_new * reciprocal_CubeSize);
			md[i].x     -= displacement;
			md[i].x_new -= displacement;
		}
		if (md[i].y_new >= CubeSize || md[i].y_new < 0.0) {
			displacement = CubeSize * FLOOR(md[i].y_new * reciprocal_CubeSize);
			md[i].y     -= displacement;
			md[i].y_new -= displacement;
		}
		if (md[i].z_new >= CubeSize || md[i].z_new < 0.0) {
			displacement = CubeSize * FLOOR(md[i].z_new * reciprocal_CubeSize);
			md[i].z     -= displacement;
			md[i].z_new -= displacement;
		}

		// Velocity Prediction
		md[i].vx_new = md[i].vx + c3*md[i].ax - c4*md[i].ax_old;
		md[i].vy_new = md[i].vy + c3*md[i].ay - c4*md[i].ay_old;
		md[i].vz_new = md[i].vz + c3*md[i].az - c4*md[i].az_old;
}

for (i=0; i<n_particles; i++) {
		count=0;
		do {
			flag = 1;
			// Velocity Prediction
			vx_prediction = md[i].vx_new;
			vy_prediction = md[i].vy_new;
			vz_prediction = md[i].vz_new;

			if (count == 0) {
				// Calculate acceleration at t+dT, based on a given velocity prediction.
				acceleration(i, md, &a, variable_size_factor, 2);
				md[i].ax_new = a.x;
				md[i].ay_new = a.y;
				md[i].az_new = a.z;
			} else {
				md[i].ax_new += g_coeff5 * (vx_prediction - vx_prediction_old);
				md[i].ay_new += g_coeff5 * (vy_prediction - vy_prediction_old);
				md[i].az_new += g_coeff5 * (vz_prediction - vz_prediction_old);
			}

			// Corrected Velocity
			md[i].vx_new = md[i].vx + c5*md[i].ax_new + c6*md[i].ax - c7*md[i].ax_old;
			md[i].vy_new = md[i].vy + c5*md[i].ay_new + c6*md[i].ay - c7*md[i].ay_old;
			md[i].vz_new = md[i].vz + c5*md[i].az_new + c6*md[i].az - c7*md[i].az_old;

			if ( FABS_SJK(vx_prediction - md[i].vx_new) > 1.0e-7*FABS_SJK(md[i].vx_new) )
				goto EXIT;

			if ( FABS_SJK(vy_prediction - md[i].vy_new) > 1.0e-7*FABS_SJK(md[i].vy_new) )
				goto EXIT;

			if ( FABS_SJK(vz_prediction - md[i].vz_new) > 1.0e-7*FABS_SJK(md[i].vz_new) )
				goto EXIT;
			continue;
```

```
		EXIT:
			flag=0;
			count++;
			vx_prediction_old = vx_prediction;
			vy_prediction_old = vy_prediction;
			vz_prediction_old = vz_prediction;

		} while (flag == 0);
	}

	// update current and old values
	for (i=0; i<n_particles; i++) {
		md[i].x_old = md[i].x;
		md[i].y_old = md[i].y;
		md[i].z_old = md[i].z;
		md[i].vx_old = md[i].vx;
		md[i].vy_old = md[i].vy;
		md[i].vz_old = md[i].vz;
		md[i].ax_old = md[i].ax;
		md[i].ay_old = md[i].ay;
		md[i].az_old = md[i].az;

		md[i].x = md[i].x_new;
		md[i].y = md[i].y_new;
		md[i].z = md[i].z_new;
		md[i].vx = md[i].vx_new;
		md[i].vy = md[i].vy_new;
		md[i].vz = md[i].vz_new;
		md[i].ax = md[i].ax_new;
		md[i].ay = md[i].ay_new;
		md[i].az = md[i].az_new;
	}
	return 0;
}


//============================================================================
// compute accelerations on the n-th particle by surrounding all the particles
//============================================================================
int acceleration(int n, MDdata *md, r_vector *a, double variable_size_factor, int mode) {
	int i, index;
	double rr, temp;
	r_vector dr;

	a->x = a->y = a->z = 0.0;
	// gravitational force + buoyant force components (negligible)
	//a->z = GRAVITY * (1 - (DENSITY_H2O/DENSITY_PROTEIN));

	if (mode == 2) {
		for (i=1; i<=g_VerletList[n][0]; i++) {		// Next Time (New Time)
			// r^2 distance at the next time
			find_rr_new(g_VerletList[n][i], n, md, variable_size_factor, &rr, &dr);
			index = min( (int)(rr * g_coeff), g_PotentialBinSize );
			temp = g_acceleration_over_r[index];		// g_coeff = (1/coeff);

			a->x += temp * dr.x;
			a->y += temp * dr.y;
			a->z += temp * dr.z;
		}
		// frictional force     component (Stokes' law)
		// g_coeff5 = -6.0*Pi()*g_R_effective*g_viscosity / g_mass;
		a->x += g_coeff5 * md[n].vx_new;
```

```
		a->y += g_coeff5 * md[n].vy_new;
		a->z += g_coeff5 * md[n].vz_new;
	}
	else if (mode == 1) {		// Curremt Time
		for (i=1; i<=g_VerletList[n][0]; i++) {		// Next Time (New Time)
			// r^2 distance at the next time
			find_rr(g_VerletList[n][i], n, md, variable_size_factor, &rr, &dr);
			// g_coeff = (1/coeff);
			index = min( (int)(rr * g_coeff), g_PotentialBinSize );
			temp = g_acceleration_over_r[index];

			a->x += temp * dr.x;
			a->y += temp * dr.y;
			a->z += temp * dr.z;
		}
		// frictional force	component (Stokes' law)
		// g_coeff5 = -6.0*Pi()*g_R_effective*g_viscosity / g_mass;
		a->x += g_coeff5 * md[n].vx;
		a->y += g_coeff5 * md[n].vy;
		a->z += g_coeff5 * md[n].vz;
	}
	else if (mode == 0) {			// Previous Time (Old Time)
		for (i=1; i<=g_VerletList[n][0]; i++) {		// Next Time (New Time)
			// r^2 distance at the next time
			find_rr_old(g_VerletList[n][i], n, md, variable_size_factor, &rr, &dr);
			// g_coeff = (1/coeff);
			index = min( (int)(rr * g_coeff), g_PotentialBinSize );
			temp = g_acceleration_over_r[index];

			a->x += temp * dr.x;
			a->y += temp * dr.y;
			a->z += temp * dr.z;
		}
		// frictional force	component (Stokes' law)
		// g_coeff5 = -6.0*Pi()*g_R_effective*g_viscosity / g_mass;
		a->x += g_coeff5 * md[n].vx_old;
		a->y += g_coeff5 * md[n].vy_old;
		a->z += g_coeff5 * md[n].vz_old;
	}

	// Brownian force component <r^2(t)> = 2Dt
	a->x += g_Brownian_Amplitude * gasdev(&g_Seed_brownian);
	a->y += g_Brownian_Amplitude * gasdev(&g_Seed_brownian);
	a->z += g_Brownian_Amplitude * gasdev(&g_Seed_brownian);

	return 0;
}


//=============================================================================
// compute forces on the n-th particle by surrounding all the particles
//=============================================================================
int force(int n, MDdata *md, r_vector *f, double variable_size_factor) {
	int i, index;
	double temp, rr;
	r_vector dr;

	f->x = f->y = f->z = 0.0;
	// gravitational force + buoyant force components (negligible)
	//f->z = mass * GRAVITY * (1 - (DENSITY_H2O/DENSITY_PROTEIN));

	for (i=1; i<=g_VerletList[n][0]; i++) {		// Next Time (New Time)
```

```
                // r^2 distance at the next time
                find_rr(g_VerletList[n][i], n, md, variable_size_factor, &rr, &dr);
                index = min( (int)(rr * g_coeff), g_PotentialBinSize );
                temp = g_Force_over_r[index];                   // g_coeff = (1/coeff);

                f->x += temp * dr.x;
                f->y += temp * dr.y;
                f->z += temp * dr.z;
        }

        // frictional force     component (Stokes' law)
        f->x += g_coeff3 * md[n].vx; // g_coeff3 = -6.0*Pi()*g_R_effective*g_viscosity;
        f->y += g_coeff3 * md[n].vy;
        f->z += g_coeff3 * md[n].vz;

        // Brownian force component <x^2(t)> = 2Dt, <r^2(t)> = 6Dt
        f->x += g_coeff4 * gasdev(&g_Seed_brownian);  // g_coeff4 = mass*g_Brownian_Amplitude;
        f->y += g_coeff4 * gasdev(&g_Seed_brownian);
        f->z += g_coeff4 * gasdev(&g_Seed_brownian);

        return 0;
}


//=============================================================================
// find spacing taking periodic boundary conditions into account
//=============================================================================
int find_rr_new(int i, int n, MDdata *md, double variable_size_factor, double *rr, r_vector *dr) {
        #ifdef CUBIC_UNIT
                double CubeSize = g_CubeSize * variable_size_factor;
                double size = CubeSize*0.5;
        #endif

        dr->x = md[n].x_new - md[i].x_new;
        dr->y = md[n].y_new - md[i].y_new;
        dr->z = md[n].z_new - md[i].z_new;

        #ifdef CUBIC_UNIT
                if (FABS_SJK(dr->x) > size)
                        dr->x -= SIGN_SJK(dr->x) * CubeSize;
                if (FABS_SJK(dr->y) > size)
                        dr->y -= SIGN_SJK(dr->y) * CubeSize;
                if (FABS_SJK(dr->z) > size)
                        dr->z -= SIGN_SJK(dr->z) * CubeSize;
        #endif

        *rr = ( (dr->x)*(dr->x) + (dr->y)*(dr->y) + (dr->z)*(dr->z) );
        return 0;
}
```

C.1.5 Levenberg-Marquardt optimization (sax_agg.c and cminpack.c)

For fitting of the protein-protein interaction potential to multiple concentration data, the Levenberg-Marquardt algorithm is applied and written in “cminpack.c”.[35, 36]

The “lmdif0” function is called for the optimization process and requires a Jacobian matrix input to calculate a gradient according the infinitesimal parameter changes. For the details of the “lmdif0” function, refer to references [35, 36].

```
//=========================================================================
// Do the Levenberg-Marquardt (LM) Optimization with Metropolis Monte Carlo (MMC), for multiple concentrations
//=========================================================================
int Multiple_Levenberg_Marquardt_Metropolis(int panel) {
	int i, j, index, info, ecode, nfev;
	int *msk, nParameter;
	double tol=(0.1*0.1), RMS_Error, RSE, *fittedArray, *coefficientArray;
	FILE *fp, *fp_r;

	g_md = (MDdata*) calloc(n_particles, sizeof(MDdata));
	// Number of Parameter + Number of different concentrations for OFFSET
	nParameter = N_PARAMETERS + (g_argc - 1);
	coefficientArray = (double*) malloc(nParameter * sizeof(double));
	fittedArray = (double*) calloc(g_argc*(N_DATA_SET-N_DATA_START), sizeof(double));
	g_logI_calc = (double*) calloc(g_argc*N_DATA_SET, sizeof(double));
	g_panel = panel;
	initialize(g_panel, g_md);

	// msk[] allows selective activation of specific parameters. '1' means enabling modifications, '0' means
	disabling modifications.
	msk = (int*) malloc(nParameter * sizeof(int));
	for (i=0; i<nParameter; i++)
		msk[i] = 1;
	if (panel == -1)
		msk[4] = 0;		// No Radius of Gyration Change

	// Initial Value of Parameters
	coefficientArray[0] = 0.1;			// Fitting Phase Constant (Offset) for the First Concentration
	coefficientArray[1] = g_PotentialWall;		// Potential Wall
	coefficientArray[2] = g_PotentialRange;		// Potential Range
	coefficientArray[3] = g_PotentialDepth;		// Potential Depth
	coefficientArray[4] = g_R_effective;		// R_Effective
	if (N_PARAMETERS == 7) {
		coefficientArray[5] = g_Repulsive_Potential_Range;	// Repulsive Potential Range
		coefficientArray[6] = g_Repulsive_Potential_Depth;	// Repulsive Potential Depth
	}
	for (i=N_PARAMETERS; i<nParameter; i++)
		coefficientArray[i] = 0.1;	// Fitting Phase Constant (Offset) for other concentrations

	if (panel <= -2) {
		g_isD0Fixed = 1;
		msk[1] = 0;
		g_PW = g_PotentialWall / (2.0*g_R_effective);
		coefficientArray[1] = (2.0*g_R_effective) * g_PW;
	}
	if (panel == -3)
		msk[2] = 0;

	// solve system (calls lmdif0 in cminpack.c)
	ecode = lmdif0(Multiple_LM_Jacobian, g_argc*(N_DATA_SET-N_DATA_START), nParameter,
	coefficientArray, msk, fittedArray, tol, &info, &nfev);

	RSE = enorm(g_argc*(N_DATA_SET-N_DATA_START), fittedArray);
	RMS_Error = sqrt( (RSE*RSE) / (g_argc*(N_DATA_SET-N_DATA_START)) );
```

```
	if (g_isD0Fixed) {
		coefficientArray[1] = (2.0*coefficientArray[4]) * g_PW;
	}

	printf("\nExit parameter = %d\n", info);
	printf("Final Root-Squared Error (RSE) =  %.13lf\n", RSE);
	printf("Final Root-Mean-Squared Error (RMSE) = %.13lf\n", RMS_Error);
	printf("PW=%.10lf (%.10lf A), PR=%.10lg A, PD=%.10lg kT, Rg=%.10lg A\n",
	coefficientArray[1]/(2.0*coefficientArray[4]), coefficientArray[1]*1.0e10, coefficientArray[2]*1.0e10,
	coefficientArray[3]/(BOLTZMANN_COEFF*g_Temperature), coefficientArray[4]/sqrt(5.0/3.0)*1.0e10);
	if (N_PARAMETERS == 7) {
		printf("Repulsive Potential Range (RPR)=%.10lg A, Repulsive Potential Depth (RPD)=%.10lg
		kT\n", coefficientArray[5]*1.0e10, coefficientArray[6]/(BOLTZMANN_COEFF*g_Temperature));
	}
	printf("Offset%d=%.10lf\t", 0, coefficientArray[0]);
	for (i=N_PARAMETERS; i<nParameter; i++)
		printf("Offset%d=%.10lf\t", i, coefficientArray[i]);

	printf("\n\nResiduals:\n");
	for (i=0; i<g_argc*(N_DATA_SET-N_DATA_START); i++)
		printf("fittedArray[%d]\t%lf\n", i, fittedArray[i]);

	for (index=0; index<g_argc; index++) {
		fp_r = fopen(g_InputFiles[index], "r");
		for (i=0; i<N_DATA_SET; i++) {
			fscanf(fp_r, "%lf\t%lf\n", &(g_qq_Data[i]), &(g_logI_Data[i]));
		}
		fclose(fp_r);

		fp = fopen(g_OutputFiles[index], "w");
		fprintf(fp, "%d\t%.10lf\t%.10lf\n", g_max_repeat, RSE, RMS_Error);

		for (i=0; i<N_DATA_SET; i++) {
			j = i + index*N_DATA_SET;
			fprintf(fp, "%.10f\t%.10f\n", g_qq[i], g_logI_calc[j]);
		}
		fclose(fp);
	}

	Destroy(coefficientArray);
	Destroy(msk);
	Destroy(fittedArray);
	Destroy(g_logI_calc);
	finalize(g_panel, g_md);

	return 0;
}


//===========================================================================
// Calculate a Jacobian matrix for the Levenberg-Marquardt optimization
//===========================================================================
void Multiple_LM_Jacobian(int m, int n, double a[], double a_h[], double y[], double **y_h, int *iflag) {
	int i, j, k, iindex, count_Guinier, count_Metropolis=1, result, index=0;
	// Number of Parameter + Number of different concentrations for OFFSET
	int nParameter = N_PARAMETERS + (g_argc - 1);
	double Metropolis_repeat, variable_size_factor, offset, offset_h, RMS_Error=0.0, RSE=0.0;
	double SumI_h1[N_DATA_SET], log_Iaverage_h0[N_DATA_SET], *Potential_h1;
	double SumI_h2[N_DATA_SET], log_Iaverage_h1[N_DATA_SET], *Potential_h2;
	double SumI_h3[N_DATA_SET], log_Iaverage_h2[N_DATA_SET], *Potential_h3;
	double SumI_h4[N_DATA_SET], log_Iaverage_h3[N_DATA_SET], *Potential_h4;
```

```
double SumI_h5[N_DATA_SET], log_Iaverage_h4[N_DATA_SET], *Potential_h5;
double SumI_h6[N_DATA_SET], log_Iaverage_h5[N_DATA_SET], *Potential_h6;
double step_h, qR_h[N_DATA_SET], I_h[N_DATA_SET], EFFECTIVE_FORCE_RANGE_2_h, coeff_h;
double coeff=1.0/sqrt(5.0/3.0), data;
char temp[256];
FILE *fp, *fp_log;
time_t s1, s2;
struct tm *newTime;
time_t szClock;

m /= g_argc;          // m : Measure Points for each concentration

if (g_isD0Fixed) {
        a[1] = (2.0*a[4]) * g_PW;
        a_h[1] = (2.0*a_h[4]) * g_PW;
}

g_EFFECTIVE_FORCE_RANGE_2 = EFFECTIVE_FORCE_CONST * (2.0 * a[4]);
g_EFFECTIVE_FORCE_RANGE_2 *= g_EFFECTIVE_FORCE_RANGE_2;
EFFECTIVE_FORCE_RANGE_2_h = EFFECTIVE_FORCE_CONST * (2.0 * a_h[4]);
EFFECTIVE_FORCE_RANGE_2_h *= EFFECTIVE_FORCE_RANGE_2_h;

Metropolis_repeat = n_particles * g_MP_Repeat;
g_step = a[4] * g_MP_StepSize;
step_h = a_h[4] * g_MP_StepSize;
fcn_count++;

if (g_Potential == NULL)
        g_Potential = (double*)malloc(sizeof(double)*g_PotentialBinSize);
Potential_h1 = (double*)malloc(sizeof(double)*g_PotentialBinSize);
Potential_h2 = (double*)malloc(sizeof(double)*g_PotentialBinSize);
Potential_h3 = (double*)malloc(sizeof(double)*g_PotentialBinSize);
Potential_h4 = (double*)malloc(sizeof(double)*g_PotentialBinSize);
Potential_h5 = (double*)malloc(sizeof(double)*g_PotentialBinSize);
Potential_h6 = (double*)malloc(sizeof(double)*g_PotentialBinSize);

if (N_PARAMETERS == 7) {
        PreCalculate_Potential(g_Potential, a[3], a[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, a[6], a[5]);
        PreCalculate_Potential(Potential_h1, a[3], a[2], a_h[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, a[6], a[5]);
        PreCalculate_Potential(Potential_h2, a[3], a_h[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, a[6], a[5]);
        PreCalculate_Potential(Potential_h3, a_h[3], a[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, a[6], a[5]);
        PreCalculate_Potential(Potential_h4, a[3], a[2], a[1], EFFECTIVE_FORCE_RANGE_2_h,
                &coeff_h, a[6], a[5]);
        PreCalculate_Potential(Potential_h5, a[3], a[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, a[6], a_h[5]);
        PreCalculate_Potential(Potential_h6, a[3], a[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, a_h[6], a[5]);
} else {
        PreCalculate_Potential(g_Potential, a[3], a[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, g_Repulsive_Potential_Depth, g_Repulsive_Potential_Range);
        PreCalculate_Potential(Potential_h1, a[3], a[2], a_h[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, g_Repulsive_Potential_Depth, g_Repulsive_Potential_Range);
        PreCalculate_Potential(Potential_h2, a[3], a_h[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, g_Repulsive_Potential_Depth, g_Repulsive_Potential_Range);
        PreCalculate_Potential(Potential_h3, a_h[3], a[2], a[1], g_EFFECTIVE_FORCE_RANGE_2,
                &g_coeff, g_Repulsive_Potential_Depth, g_Repulsive_Potential_Range);
        PreCalculate_Potential(Potential_h4, a[3], a[2], a[1], EFFECTIVE_FORCE_RANGE_2_h,
                &coeff_h, g_Repulsive_Potential_Depth, g_Repulsive_Potential_Range);
```

```
}

// for Multiple Files
for (index=0; index<g_argc; index++) {
	s1 = time (NULL);

	if (index == 0) {
		offset = a[0];
		offset_h = a_h[0];
	} else {
		offset = a[index-1 + N_PARAMETERS];
		offset_h = a_h[index-1 + N_PARAMETERS];
	}

	sprintf(temp, "%s.log", g_OutputFiles[index]);
	fp_log=fopen(temp, "a");

	// Get UNIX-style time and display as number and string.
	time(&szClock);
	newTime = localtime(&szClock);
	fprintf(fp_log, "%s", asctime(newTime));
	printf("%s", asctime(newTime));

	if (N_PARAMETERS == 7) {
		fprintf(fp_log, "[%d] : %s (%g, %g, %g, %g, %g, %g, %g) ->
		(%g, %g, %g, %g, %g, %g, %g)\n", fcn_count, g_OutputFiles[index], offset,
		a[1]/(2.0*a[4]), a[2], a[3]/(BOLTZMANN_COEFF*g_Temperature), a[4]*coeff, a[5],
		a[6]/(BOLTZMANN_COEFF*g_Temperature), offset_h, a_h[1]/(2.0*a[4]), a_h[2],
		a_h[3]/(BOLTZMANN_COEFF*g_Temperature), a_h[4]*coeff, a_h[5],
		a_h[6]/(BOLTZMANN_COEFF*g_Temperature));
		printf("[%d] : %s (%g, %g, %g, %g, %g, %g, %g) -> (%g, %g, %g, %g, %g, %g, %g)\n",
		fcn_count, g_OutputFiles[index], offset, a[1]/(2.0*a[4]), a[2],
		a[3]/(BOLTZMANN_COEFF*g_Temperature), a[4]*coeff, a[5],
		a[6]/(BOLTZMANN_COEFF*g_Temperature), offset_h, a_h[1]/(2.0*a[4]), a_h[2],
		a_h[3]/(BOLTZMANN_COEFF*g_Temperature), a_h[4]*coeff, a_h[5],
		a_h[6]/(BOLTZMANN_COEFF*g_Temperature));
	} else {
		fprintf(fp_log, "[%d] : %s (%g, %g, %g, %g, %g) -> (%g, %g, %g, %g, %g)\n",
		fcn_count, g_OutputFiles[index], offset, a[1]/(2.0*a[4]), a[2],
		a[3]/(BOLTZMANN_COEFF*g_Temperature), a[4]*coeff, offset_h, a_h[1]/(2.0*a[4]),
		a_h[2], a_h[3]/(BOLTZMANN_COEFF*g_Temperature), a_h[4]*coeff);
		printf("[%d] : %s (%g, %g, %g, %g, %g) -> (%g, %g, %g, %g, %g)\n", fcn_count,
		g_OutputFiles[index], offset, a[1]/(2.0*a[4]), a[2],
		a[3]/(BOLTZMANN_COEFF*g_Temperature), a[4]*coeff, offset_h, a_h[1]/(2.0*a[4]),
		a_h[2], a_h[3]/(BOLTZMANN_COEFF*g_Temperature), a_h[4]*coeff);
	}
	fclose(fp_log);

	fp = fopen(g_InputFiles[index], "r");
	for (i=0; i<N_DATA_SET; i++) {
		g_SumI[i] = 0.0;
		SumI_h1[i] = 0.0;
		SumI_h2[i] = 0.0;
		SumI_h3[i] = 0.0;
		SumI_h4[i] = 0.0;
		SumI_h5[i] = 0.0;
		SumI_h6[i] = 0.0;

		fscanf(fp, "%lf\t%lf\n", &(g_qq_Data[i]), &(g_logI_Data[i]));

		g_qR[i] = g_q_Data[i] * a[4];
		g_I[i] = 3.0 * (sin(g_qR[i]) - g_qR[i]*cos(g_qR[i])) / (g_qR[i]*g_qR[i]*g_qR[i]);
```

```
            g_I[i] = (g_I[i]*g_I[i]) / (n_particles);

            qR_h[i] = g_q_Data[i] * a_h[4];
            I_h[i] = 3.0 * (sin(qR_h[i]) - qR_h[i]*cos(qR_h[i])) / (qR_h[i]*qR_h[i]*qR_h[i]);
            I_h[i] = (I_h[i]*I_h[i]) / (n_particles);
}
fclose(fp);

count_Guinier = 1;
variable_size_factor = Random_Distribution(g_panel, g_md, g_CubeSizeS[index]);

while ( (g_Running == 1) && (count_Guinier <= g_max_repeat) ) {
            for (count_Metropolis=1; count_Metropolis<=Metropolis_repeat; count_Metropolis++)
                        result = LM_Metropolis_Sampling(g_panel, g_md, variable_size_factor,
                        (int)NR_Random(0.0, n_particles, &g_Seed_MC2), g_Potential, g_step,
                        g_EFFECTIVE_FORCE_RANGE_2, g_coeff, g_CubeSizeS[index]);
            LM_SAXS_Guinier(g_panel, &count_Guinier, g_md, variable_size_factor, g_SumI,
            g_log_Iaverage, g_I, g_OutputFiles[index], g_CubeSizeS[index]);

            if (fcn_count > 1) {
                        // (1) Potential Wall
                        for (count_Metropolis=1; count_Metropolis<=20*n_particles;
                        count_Metropolis++)
                                    result = LM_Metropolis_Sampling(g_panel, g_md,
                                    variable_size_factor, (int)NR_Random(0.0, n_particles,
                                    &g_Seed_MC2), Potential_h1, g_step,
                                    g_EFFECTIVE_FORCE_RANGE_2, g_coeff, g_CubeSizeS[index]);
                        LM_SAXS_Guinier(g_panel, &count_Guinier, g_md, variable_size_factor,
                        SumI_h1, log_Iaverage_h0, g_I, g_OutputFiles[index], g_CubeSizeS[index]);

                        // (2) Potential Range
                        for (count_Metropolis=1; count_Metropolis<=20*n_particles;
                        count_Metropolis++)
                                    result = LM_Metropolis_Sampling(g_panel, g_md,
                                    variable_size_factor, (int)NR_Random(0.0, n_particles,
                                    &g_Seed_MC2), Potential_h2, g_step,
                                    g_EFFECTIVE_FORCE_RANGE_2, g_coeff, g_CubeSizeS[index]);
                        LM_SAXS_Guinier(g_panel, &count_Guinier, g_md, variable_size_factor,
                        SumI_h2, log_Iaverage_h1, g_I, g_OutputFiles[index], g_CubeSizeS[index]);

                        // (3) Potential Depth
                        for (count_Metropolis=1; count_Metropolis<=20*n_particles;
                        count_Metropolis++)
                                    result = LM_Metropolis_Sampling(g_panel, g_md,
                                    variable_size_factor, (int)NR_Random(0.0, n_particles,
                                    &g_Seed_MC2), Potential_h3, g_step,
                                    g_EFFECTIVE_FORCE_RANGE_2, g_coeff, g_CubeSizeS[index]);
                        LM_SAXS_Guinier(g_panel, &count_Guinier, g_md, variable_size_factor,
                        SumI_h3, log_Iaverage_h2, g_I, g_OutputFiles[index], g_CubeSizeS[index]);

                        // (4) R_Effective
                        for (count_Metropolis=1; count_Metropolis<=20*n_particles;
                        count_Metropolis++)
                                    result = LM_Metropolis_Sampling(g_panel, g_md,
                                    variable_size_factor, (int)NR_Random(0.0, n_particles,
                                    &g_Seed_MC2), Potential_h4, step_h,
                                    EFFECTIVE_FORCE_RANGE_2_h, coeff_h, g_CubeSizeS[index]);
                        LM_SAXS_Guinier(g_panel, &count_Guinier, g_md, variable_size_factor,
                        SumI_h4, log_Iaverage_h3, I_h, g_OutputFiles[index], g_CubeSizeS[index]);

                        if (N_PARAMETERS == 7) {
                                    // (5) Repulsive Potential Range
```

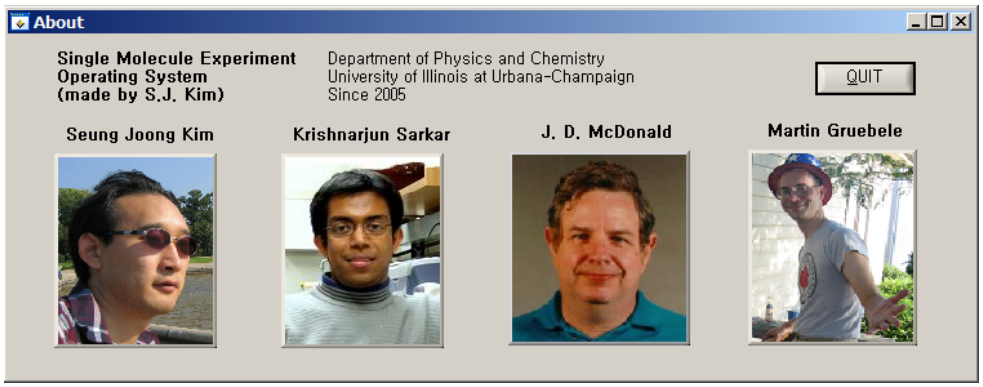
```
                                for (count_Metropolis=1; count_Metropolis<=20*n_particles;
                                count_Metropolis++)
                                        result = LM_Metropolis_Sampling(g_panel, g_md,
                                        variable_size_factor, (int)NR_Random(0.0, n_particles,
                                        &g_Seed_MC2), Potential_h5, g_step,
                                        g_EFFECTIVE_FORCE_RANGE_2, g_coeff,
                                        g_CubeSizeS[index]);
                                LM_SAXS_Guinier(g_panel, &count_Guinier, g_md,
                                variable_size_factor, SumI_h5, log_Iaverage_h4, g_I,
                                g_OutputFiles[index], g_CubeSizeS[index]);

                                // (6) Repulsive Potential Depth
                                for (count_Metropolis=1; count_Metropolis<=20*n_particles;
                                count_Metropolis++)
                                        result = LM_Metropolis_Sampling(g_panel, g_md,
                                        variable_size_factor, (int)NR_Random(0.0, n_particles,
                                        &g_Seed_MC2), Potential_h6, g_step,
                                        g_EFFECTIVE_FORCE_RANGE_2, g_coeff,
                                        g_CubeSizeS[index]);
                                LM_SAXS_Guinier(g_panel, &count_Guinier, g_md,
                                variable_size_factor, SumI_h6, log_Iaverage_h5, g_I,
                                g_OutputFiles[index], g_CubeSizeS[index]);
                        }
                }
                variable_size_factor = Random_Distribution(g_panel, g_md, g_CubeSizeS[index]);
                count_Guinier++;
        }
        RSE = 0.0;
        for (i=0; i<m; i++) {
                j = i + index*m;
                iindex = N_DATA_START + i;
                data = g_logI_Data[iindex];

                y[j] = (g_log_Iaverage[iindex] + offset) - data;
                RSE += y[j]*y[j];   // Calculate Squared Error (SE)

                y_h[0][j] = y[j];
                if (fcn_count > 1) {
                        y_h[1][j] = (log_Iaverage_h0[iindex] + offset) - data;
                        y_h[2][j] = (log_Iaverage_h1[iindex] + offset) - data;
                        y_h[3][j] = (log_Iaverage_h2[iindex] + offset) - data;
                        y_h[4][j] = (log_Iaverage_h3[iindex] + offset) - data;
                }
                else {
                        y_h[1][j] = y_h[2][j] = y_h[3][j] = y_h[4][j] = y[j];
                }
                if (N_PARAMETERS == 7) {
                        if (fcn_count > 1) {
                                y_h[5][j] = (log_Iaverage_h4[iindex] + offset) - data;
                                y_h[6][j] = (log_Iaverage_h5[iindex] + offset) - data;
                        } else {
                                y_h[5][j] = y_h[6][j] = y[j];
                        }
                }

                for (k=N_PARAMETERS; k<nParameter; k++)
                        y_h[k][j] = y[j];

                if (index == 0)
                        y_h[0][j] += offset_h - offset;
                else
                        y_h[index-1 + N_PARAMETERS][j] += offset_h - offset;
```

```
                }
                RSE = sqrt(RSE);   // Calculate Root Squared Error (RSE)
                RMS_Error = sqrt( (RSE*RSE) / m );   // Calculate Root Mean Squared Error (RMSE)

                sprintf(temp, "%s.log", g_OutputFiles[index]);
                fp_log = fopen(temp, "a");
                fp=fopen(g_OutputFiles[index], "w");
                fprintf(fp, "%d\t%.10lf\t%.10lf\n", count_Guinier-1, RSE, RMS_Error);
                for (i=0; i<N_DATA_SET; i++) {
                        fprintf(fp, "%.10f\t%.10f\n", g_qq[i], (g_log_Iaverage[i] + offset));
                        j = i + index*N_DATA_SET;
                        g_logI_calc[j] = g_log_Iaverage[i] + offset;
                }

                fprintf(fp, "RSE = %.13lf\t\tRMSE = %.13lf\n", RSE, RMS_Error);
                fprintf(fp_log, "RSE = %.13lf\t\tRMSE = %.13lf\n", RSE, RMS_Error);
                printf("RSE = %.13lf\t\tRMSE = %.13lf\n", RSE, RMS_Error);

                s2 = time (NULL);
                fprintf(fp, "Execution Time = %g mins\n\n", (s2-s1)/60.0);
                fprintf(fp_log, "Execution Time = %g mins\n\n", (s2-s1)/60.0);
                printf("Execution Time = %g mins\n\n", (s2-s1)/60.0);

                fclose(fp);
                fclose(fp_log);
        }

        free(Potential_h1);
        free(Potential_h2);
        free(Potential_h3);
        free(Potential_h4);
        free(Potential_h5);
        free(Potential_h6);
}


//=============================================================================
// Distribute proteins according to a potential model, for Multiple Concentration (with Levenberg-marquardt)
//=============================================================================
int LM_Metropolis_Sampling(int panel, MDdata *md, double variable_size_factor, int index, double *Potential, double
step, double EFFECTIVE_FORCE_RANGE_2, double coeff, double iCubeSize) {
        double energy1=0.0, energy2=0.0, delta_U, tempx, tempy, tempz;
        double CubeSize, u=0.0;
        int j;

        double rr, reciprocal_CubeSize;
        r_vector dr;

        CubeSize = iCubeSize * variable_size_factor;
        reciprocal_CubeSize = 1.0/CubeSize;

        tempx = NR_Random(-step, step, &g_Seed_xyz);
        tempy = NR_Random(-step, step, &g_Seed_xyz);
        tempz = NR_Random(-step, step, &g_Seed_xyz);

        for (j=0; j<n_particles; j++) {
                if (j != index) {
                        find_rr(index, j, md, variable_size_factor, &rr, &dr);

                        if (rr < EFFECTIVE_FORCE_RANGE_2)  // using nearest separation rule
                                energy1 += Potential[(int)(rr * coeff)];
                }
```

```
	}

	md[index].x_old = md[index].x;
	md[index].y_old = md[index].y;
	md[index].z_old = md[index].z;

	md[index].x += tempx;
	md[index].y += tempy;
	md[index].z += tempz;

	// Periodic Boundary Condition
	if (md[index].x >= CubeSize || md[index].x < 0.0 )
		md[index].x -= CubeSize * FLOOR(md[index].x * reciprocal_CubeSize);
	if (md[index].y >= CubeSize || md[index].y < 0.0)
		md[index].y -= CubeSize * FLOOR(md[index].y * reciprocal_CubeSize);
	if (md[index].z >= CubeSize || md[index].z < 0.0)
		md[index].z -= CubeSize * FLOOR(md[index].z * reciprocal_CubeSize);

	for (j=0; j<n_particles; j++) {
		if (j != index) {
			find_rr(index, j, md, variable_size_factor, &rr, &dr);

			if (rr < EFFECTIVE_FORCE_RANGE_2)  // using nearest separation rule
				energy2 += Potential[(int)(rr * coeff)];
		}
	}
	delta_U = energy2 - energy1;

	if (delta_U >= 0.0) {
		// Rollback; Rejected
		if ( NR_Random(0.0, 1.0, &g_Seed_MC) > exp( delta_U * g_reciprocal_of_KT ) ) {
			md[index].x = md[index].x_old;
			md[index].y = md[index].y_old;
			md[index].z = md[index].z_old;

			return -1;
		}
	}
	return 0;
}


//=========================================================================
// Calculate and average SAXS scattering profiles, in Guinier plot for Levenberg-Marquardt Method
//=========================================================================
int LM_SAXS_Guinier(int panel, int *count_Guinier, MDdata *md, double variable_size_factor, double *SumI, double
*log_Iaverage, double *I, char *OutputFile, double iCubeSize) {
	int i;

	for (i=0; i<N_DATA_SET; i++)
		SumI[i] += Calc_Scattering_Intensity(i, md, variable_size_factor, iCubeSize);

	if (*count_Guinier - (int)(*count_Guinier/g_SAXS_MP_rate)*g_SAXS_MP_rate == 0) {
		//FILE *fp;
		//fp=fopen(OutputFile, "w");
		//fprintf(fp, "%d\n", *count_Guinier);

		for (i=0; i<N_DATA_SET; i++) {
			log_Iaverage[i] = log( I[i] * SumI[i] / (double)(*count_Guinier) * g_volume_correction );
			//fprintf(fp, "%.10f\t%.10f\n", g_qq[i], log_Iaverage[i]);
		}
		//fclose(fp);
```

```
	}
	return 0;
}
```

C.2 ASMOS (The automated single molecule operating system)



C.2.1 Time calibration module (sm_fn.c)

```
//=============================================================================================
// It prepares to convert raw data to the uniformly corrected one by generating conversion tables for the white light source.
//=============================================================================================
int TimeCalibration(void) {
	int hist[1024], hist1[1024], hist2[1024], hist3[1024], hist4[1024], hist5[1024], hist6[1024], hist0[1024];
	int i, err, index, laser_tick, PMTBuffer, decay;
	double decay_corrected;
	char temp[50];
	FILE *in, *out;

	for (i = 0; i < 1024; i++) {
		hist[i] = 0;
		hist1[i] = 0;
		hist2[i] = 0;
		hist3[i] = 0;
		hist4[i] = 0;
		hist5[i] = 0;
		hist6[i] = 0;
		hist0[i] = 0;

		gCorrections[i] = NULL;
		gCorrections1[i] = NULL;
		gCorrections2[i] = NULL;
		gCorrections3[i] = NULL;
		gCorrections4[i] = NULL;
		gCorrections5[i] = NULL;
		gCorrections6[i] = NULL;
```

```
		gDirectCorrection[i] = 0.0;
		gDirectCorrection1[i] = 0.0;
		gDirectCorrection2[i] = 0.0;
		gDirectCorrection3[i] = 0.0;
		gDirectCorrection4[i] = 0.0;
		gDirectCorrection5[i] = 0.0;
		gDirectCorrection6[i] = 0.0;
}

// Open a calibration histogram file
in = fopen("SM_data_calibration.his","r");
if (in != NULL) {
		i=0;
		while ( err = fscanf (in, "%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", &index, &hist[i], &hist1[i],
		&hist2[i], &hist3[i], &hist4[i], &hist5[i], &hist6[i]) > 0 )
			i++;
		fclose(in);
// if there is no calibration histogram file
} else {
		in = fopen("SM_data_calibration.txt","r");
		if (in != NULL) {
			// Ignoring the first row showing date/time information
			sprintf(temp, "[%s, %s]\n", DateStr(), TimeStr());
			fseek(in, strlen(temp), SEEK_SET);

			// Making a Histogram
			while ( err = fscanf (in, "%d\t%d\t%d\t%lf", &laser_tick, &PMTBuffer, &decay,
			&decay_corrected) > 0 ) {
				switch ( PMTBuffer ) {
					case 1 :   hist1[decay]++;
						   break;
					case 2 :   hist2[decay]++;
						   break;
					case 3 :   hist3[decay]++;
						   break;
					case 4 :   hist4[decay]++;
						   break;
					case 5 :   hist5[decay]++;
						   break;
					case 6 :   hist6[decay]++;
						   break;
					default :  hist0[decay]++;
						   break;
				}
				hist[decay]++;
			}
			fclose(in);
		} else {
			MessagePopup("Warning", "Please do the Time-Calibration !");
			return 0;
		}
		// Saving a calibration histogram file
		out = fopen("SM_data_calibration.his","w");
		for (i = 0; i < 1024; i++)
			fprintf(out, "%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", i, hist[i], hist1[i], hist2[i], hist3[i],
			hist4[i], hist5[i], hist6[i]);
		fclose(out);
}
// gTimingStep : default 555 -> 11.1 ns (90Mhz), which means each step corresponds to 20 pico second.
GetCtrlVal(SM, PANEL_NUMERIC_TIMING_STEP, &gTimingStep);
```

```
	// 555 -> 11.1 ns (90Mhz), which means each step corresponds to 20 pico second.
	build_corrections(hist, 0, 1024, gTimingStep);
	build_corrections_eachPMT(hist1, 0, 1024, gTimingStep, gCorrections1, gDirectCorrection1);
	build_corrections_eachPMT(hist2, 0, 1024, gTimingStep, gCorrections2, gDirectCorrection2);
	build_corrections_eachPMT(hist3, 0, 1024, gTimingStep, gCorrections3, gDirectCorrection3);
	build_corrections_eachPMT(hist4, 0, 1024, gTimingStep, gCorrections4, gDirectCorrection4);
	build_corrections_eachPMT(hist5, 0, 1024, gTimingStep, gCorrections5, gDirectCorrection5);
	build_corrections_eachPMT(hist6, 0, 1024, gTimingStep, gCorrections6, gDirectCorrection6);
	return 0;
}


//=============================================================================================
// It builds correction tables for each PMT.  Originally written by Dr. McDonalds, and updated by SJ Kim.
//=============================================================================================
void build_corrections_eachPMT(int *hist, int first, int ndata, int nbins, partials **Corrections, double
*DirectCorrection) {
	int i, j, k, m, q;
	partials temp[500];
	double target, target2, residual;

	if (nbins > 1024) {
		MessagePopup("Error", "Error Code : E01");
		return;
	}

	for (i = 0; i < 1024; i++)
		if (Corrections[i])
			free(Corrections[i]);

	k = 0;
	for (i = first; i < first + ndata; i++)
		k += hist[i];

	target = (double)k / (double)nbins;
	m = first;
	residual = (double)hist[m];

	// Building Corrections array for statistical correction of original decay time by using Histogram...
	for(i=0; i<nbins; i++) {
		j = 0;
		target2 = target;

		do {
			if (residual > 0.0) {
				if (residual < target2) {
					temp[j].index = m;
					temp[j].fraction = (double)residual / (double)hist[m];
					target2 -= residual;
					residual = (double)hist[++m];
				} else {
					temp[j].index = m;
					temp[j].fraction = (double)target2 / (double)hist[m];
					residual -= target2;
					target2 = 0.0;
				}
				if (++j > 498) {
					MessagePopup("Error", "Error Code : E02");
					return;
				}
			} else {
				if (++m >= ndata)
```

```
                                        break;
                                residual = (double)hist[m];
                        }
                } while (target2 > 0.0);

                temp[j].index = -1.0;
                temp[j].fraction = 0.0;

                if (++j > 499) {
                        MessagePopup("Error", "Error Code : E03");
                        return;
                }

                Corrections[i] = (partials *) malloc( j * sizeof(partials) );
                for (q = 0; q < j; q++) {
                        Corrections[i][q].index = temp[q].index;
                        Corrections[i][q].fraction = temp[q].fraction;
                }
        }

        // Building DirectCorrection array For Direct 1:1 correspondence between original & corrected decay time
        j=0;
        k=0;
        for (i=0; i<ndata; i++) {       // ndata : 1024
                while (Corrections[j][k].index == i) {
                        DirectCorrection[i] += Corrections[j][k].fraction * j;
                        k++;
                        if (Corrections[j][k].index == -1) {
                                k=0;
                                j++;
                                if (j >= nbins)         return;
                        }
                }
        }
}


//==============================================================================
// It builds a correction table for total photon counts.  Originally written by Dr. McDonalds, and updated by SJ Kim.
//==============================================================================
void build_corrections(int *hist, int first, int ndata, int nbins) {
        int i, j;
        int k, m, q;
        partials temp[500];
        double target, target2, residual;

        if (nbins > 1024) {
                MessagePopup("Error", "Error Code : E04");
                return;
        }

        for (i = 0; i < 1024; i++)
                if (gCorrections[i])
                        free(gCorrections[i]);

        k = 0;
        for (i = first; i < first + ndata; i++)
                k += hist[i];

        target = (double)k / (double)nbins;
        m = first;
        residual = (double)hist[m];
```

```
	// Building gCorrections array for statistical correction of original decay time by using Histogram...
	for(i=0; i<nbins; i++) {
		j = 0;
		target2 = target;

		do {
			if (residual > 0.0) {
				if (residual < target2) {
					temp[j].index = m;
					temp[j].fraction = (double)residual / (double)hist[m];
					target2 -= residual;
					residual = (double)hist[++m];
				} else {
					temp[j].index = m;
					temp[j].fraction = (double)target2 / (double)hist[m];
					residual -= target2;
					target2 = 0.0;
				}
				if (++j > 498) {
					MessagePopup("Error", "Error Code : E05");
					return;
				}
			} else {
				if (++m >= ndata)
					break;
				residual = (double)hist[m];
			}
		} while (target2 > 0.0);

		temp[j].index = -1.0;
		temp[j].fraction = 0.0;

		if (++j > 499) {
			MessagePopup("Error", "Error Code : E06");
			return;
		}

		gCorrections[i] = (partials *) malloc( j * sizeof(partials) );
		for (q = 0; q < j; q++) {
			gCorrections[i][q].index = temp[q].index;
			gCorrections[i][q].fraction = temp[q].fraction;
		}
	}

	// Building gDirectCorrection array For Direct 1:1 correspondence between original & corrected decay time
	j=0;
	k=0;
	for (i=0; i<ndata; i++) {	// ndata : 1024
		while (gCorrections[j][k].index == i) {
			gDirectCorrection[i] += gCorrections[j][k].fraction * j;
			k++;
			if (gCorrections[j][k].index == -1) {
				k=0;
				j++;
				if (j >= nbins)		return;
			}
		}
	}
}
```

```
//==============================================================================
// It converts the raw data to the uniformly corrected one per each PMT
//==============================================================================
void correct_eachPMT(int *data, double *decay, int first, int ndata, int nbins, partials **Corrections) {
	int i, j;

	if (nbins > 1024) {
		MessagePopup("Error", "Error Code : E07");
		return;
	}

	for (i=0; i<nbins; i++) {
		j = 0;
		decay[i] = 0.0;
		while (Corrections[i][j].index != -1.0) {
			decay[i] += data[ Corrections[i][j].index ] * Corrections[i][j].fraction;
			j++;
		}
	}
}


//==============================================================================
// It converts the raw data to the uniformly corrected one for the total photon counts
//==============================================================================
void correct(int *data, double *decay, int first, int ndata, int nbins) {
	int i, j;

	if (nbins > 1024) {
		MessagePopup("Error", "Error Code : E08");
		return;
	}

	for (i=0; i<nbins; i++) {
		j = 0;
		decay[i] = 0.0;
		while (gCorrections[i][j].index != -1.0) {
			decay[i] += data[ gCorrections[i][j].index ] * gCorrections[i][j].fraction;
			j++;
		}
	}
}


//==============================================================================
// It converts the raw data to the uniformly corrected one per each PMT, with an offset (a shift).
//==============================================================================
void correct_with_offset(int *data, double *decay, int first, int ndata, int nbins, partials **Corrections, int offset) {
	int i, j;
	double *decay_temp;

	decay_temp = (double*) malloc(nbins*sizeof(double));

	if (nbins > 1024) {
		MessagePopup("Error", "Error Code : E09");
		return;
	}

	for (i=0; i<nbins; i++) {
		j = 0;
		decay_temp[i] = 0.0;
		while (Corrections[i][j].index != -1.0) {
```

```
                        decay_temp[i] += data[ Corrections[i][j].index ] * Corrections[i][j].fraction;
                        j++;
                }
        }

        for (i=0; i<offset; i++)
                decay[nbins-offset+i] = decay_temp[i];

        for (i=offset; i<nbins; i++)
                decay[i-offset] = decay_temp[i];

        free(decay_temp);
}
```

C.2.2 Droplet generation module (sm_fn.c)

```
//=============================================================================================
// Initialize the droplet generator. Characters in the parenthesis indicate the protocol index.
//=============================================================================================
int Initial_Connection(void) {
        SendCommand(JetDrv, MFJDRV_RESET);                      // 1. Soft Reset (01)
        SendCommand(JetDrv, MFJDRV_GETVERSION);                 // 2. Get version (F0)
        SendCommand(JetDrv, MFJDRV_GETCHANNEL);                 // 2.5. Get number of channels (0D)

        return 0;
}


//=============================================================================================
// Start droplet generation. Characters in the parenthesis indicate the protocol index.
//=============================================================================================
int Droplet_Start(void) {
        SendCommand(JetDrv, MFJDRV_PULSE);                      // 3. Set pulse wave form (06)
        SendCommand(JetDrv, MFJDRV_CONTMODE);                   // 4. Set trigger mode (04)
        SendCommand(JetDrv, MFJDRV_DROPS);                      // 5. Set drops/trigger (03)
        SendCommand(JetDrv, MFJDRV_FULLFREQ);                   // 6. Set frequency (12)
        SendCommand(JetDrv, MFJDRV_STROBEDIV);                  // 7. Set strobe divider (07)
        SendCommand(JetDrv, MFJDRV_STROBEENABLE);               // 8. Strobe Enable (10)
        SendCommand(JetDrv, MFJDRV_STROBEDELAY);                // 9. Set Strobe delay (13)
        SendCommand(JetDrv, MFJDRV_SOURCE);                     // 10. Set trigger source (08)
        SendCommand(JetDrv, MFJDRV_SOFTTRIGGER);                // 11. Trigger Output (09) "START"

        WriteDropletParamaters();

        return 0;
}


//=============================================================================================
// Stop droplet generation.
//=============================================================================================
int Droplet_Stop(void) {
        int temp = gJets[gCJ].fMode;

        gJets[gCJ].fMode = 0;
        SendCommand(JetDrv, MFJDRV_CONTMODE);                   // 4. Set trigger mode (04) to "Single"
        Delay(0.5);                                             // Wait for the pulse to stop
```

```
	gJets[gCJ].fMode = temp;

	// by SJ Kim at 07/24/2007, for the ultimate droplet stop.
	if ( SerialPort_Connection() == 0 ) {				// Initialize the serial port connection.
		Initial_Connection();					// Initialize the droplet generator
		MessagePopup("Droplet Stop", "The droplet generator has stopped successfully.");
	} else
		MessagePopup("Droplet Stop", "There is an error during serial port connection.");
	return 0;
}


//==============================================================================
// by SJ Kim at 11/07/2007, for the dynamic parameter change of the droplet generator.
//==============================================================================
int Droplet_Update(void) {
	int temp = gJets[gCJ].fMode;

	gJets[gCJ].fMode = 0;
	SendCommand(JetDrv, MFJDRV_CONTMODE);		// 4. Set trigger mode (04) to "Single"
	Delay(0.5);					// Wait for the pulse to stop (> minimal dead time (~60 ms))

	gJets[gCJ].fMode = temp;			// Roll-back mode value as before

	Droplet_Start();				// Update Droplet Information & Re-Start
	return 0;
}
```

C.2.3 Data acquisition module (sm_fn.c)

```
//==============================================================================
// Start “Trap & Data acquisition”.  It executes the data acquisition function (StartTrapDAQ_Thread()), and start trigger.
//==============================================================================

int CVICALLBACK StartTrapDAQCallback (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event)
		{
		case EVENT_COMMIT:
			SetCtrlAttribute (panelHandle, PANEL_START_TRAP_DAQ, ATTR_DIMMED, 1);
			StartTrapDAQ_Thread (panelHandle, panelParameter2);
		#ifdef CONTINUOUS_TRAP
			Sleep(4000);
		#endif
			SetCtrlAttribute (panelHandle, PANEL_STOP_TRAP_DAQ, ATTR_DIMMED, 0);
			SetCtrlVal (panelHandle, PANEL_LED, 1);
			ProcessDrawEvents();
		#ifdef CONTINUOUS_TRAP
			trigger();
		#endif
			break;
		}
	return 0;
}
```

```
//==============================================================================
// Stop “Trap & Data acquisition”
//==============================================================================
int CVICALLBACK StopTrapDAQCallback (int panel, int control, int event,
		void *callbackData, int eventData1, int eventData2)
{
	switch (event)
		{
		case EVENT_COMMIT:
			SetCtrlAttribute (panelHandle, PANEL_STOP_TRAP_DAQ, ATTR_DIMMED, 1);
			KillEverything();
			SetCtrlAttribute (panelHandle, PANEL_START_TRAP_DAQ, ATTR_DIMMED, 0);
			DeleteGraphPlot(panelHandle, PANEL_GRAPH, -1, VAL_IMMEDIATE_DRAW);
			SetCtrlVal (panelHandle, PANEL_LED, 0);
			ProcessDrawEvents();
			SetgTrapDAQRunning(0);

			break;
		case EVENT_RIGHT_CLICK:

			break;
		}
	return 0;
}


//==============================================================================
// Current Version for Data Acquisition Thread
//==============================================================================
int StartTrapDAQ_Thread(int panelHandle, int panelPara) {
	bool32 done=0;
	char AOPhysical[20]={'\0'};
	char errBuff[2048] ={'\0'};
	char chan[256];
	char REQclockSource6534[256]={“/Dev2/PFI2”};    // REQ Signal from Doug's Box, into 6534 (port4/line2)
	double amplitude, phase=0.0, min=-10.0, max=10.0;
	double AOfrequency, AOrate, *AOdata=NULL;
	double resolution, width, peak_width, delay, rise, peak_rise, rate;
	double time12;
	double LaserFrequency;
	int DAQmxError = DAQmxSuccess;
	int i, written, bufferSize, sampsPerCycle, status, bIsAutoCalibration_Checked;
	unsigned int sampsToRead;
	unsigned int *RESETdata=NULL;
	char output_file[256];
	FILE *fp;

	InitializeCriticalSection( &gLock_READ );
	InitializeCriticalSection( &gLock_WRITE );

QueryPerformanceCounter(&gTime1);
QueryPerformanceFrequency (&gTicksPerSecond);

	// Load PMT offset information
	if ( (fp = fopen("PMT_Calibration.txt","r")) != NULL ) {
		status = fscanf (fp, "%d %d %d %d %d %d", &gPMT1_Offset, &gPMT2_Offset, &gPMT3_Offset,
		&gPMT4_Offset, &gPMT5_Offset, &gPMT6_Offset);
		if (status < 6) {
			MessagePopup("error", "\"PMT_Calibration.txt\" doesn't have all parameters.");
			return -1;
		}
	} else {
```

```
                MessagePopup("error", "\"PMT_Calibration.txt\" doesn't exist.");
                return -2;
        }
        fclose(fp);

        Init_Histrograms();

        // Set up overlapped I/O structure fields.
        ZeroMemory( &gOverlapped, sizeof(gOverlapped) );
        gOverlapped.hEvent = CreateEvent(NULL, TRUE, TRUE, NULL);
        gFileSize.QuadPart = 0;

        GetCtrlVal(AUTO_CALIBRATION, PANEL_CALI_CHECKBOX_AUTO, &bIsAutoCalibration_Checked);
        if (bIsAutoCalibration_Checked)
                sprintf(output_file, "H:\\output1.dat");
        else
                sprintf(output_file, "H:\\output.dat");

        // File Pointer Configuration
        ghFile = CreateFile( output_file, GENERIC_WRITE, 0, NULL,
#ifdef OVERLAPPED_IO
                CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, NULL );
#else
                CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );
#endif
        gFp = fopen("D:\\SM_photon_density_Log.txt", "w");
        gFp_thread = fopen("D:\\SM_timing_Log.txt", "w");

        // Delete any already running timer
        if (g_AutoCalibrationTimerId > 0) {
                double width;
                GetCtrlVal(AUTO_CALIBRATION, PANEL_CALI_NUMERIC_CALI_DURATION, &width);

                SetAsyncTimerAttribute(g_AutoCalibrationTimerId, ASYNC_ATTR_ENABLED, 0); // < 0.01 ms
                DiscardAsyncTimer (g_AutoCalibrationTimerId);
                g_AutoCalibrationTimerId = 0;
                Delay(width + 1.0);
        }
        // Stop Autocalibration task
        if ( gAutoCalibration_task != 0 ) {
                DAQmxStopTask(gAutoCalibration_task);
                DAQmxClearTask(gAutoCalibration_task);
                gAutoCalibration_task = 0;
        }

        // Reset Devices
        DAQmxResetDevice ("Dev1");
        DAQmxResetDevice ("Dev2");

        // For Analog Output Laser Trapping
        GetCtrlVal(panelPara, PANELPARA2_AOPHYSICAL, AOPhysical);
        WriteCharParameter("AOPhysical", AOPhysical);
        GetCtrlVal(panelPara, PANELPARA2_RESOLUTION, &resolution);              resolution *= 1.0e-6;
        WriteParameter("AOresolution", (double)resolution);
        GetCtrlVal(panelPara, PANELPARA2_RISE, &rise);                          amplitude = rise / 2.0;
        WriteParameter("AOrise", (double)rise);
        GetCtrlVal(panelPara, PANELPARA2_WIDTH, &width);                        width *= 1.0e-6;
        WriteParameter("AOwidth", (double)width);
        GetCtrlVal(panelPara, PANELPARA2_FREQUENCY, &AOfrequency);
        WriteParameter("AOfrequency", (double)AOfrequency);
        GetCtrlVal(panelPara, PANELPARA2_PEAK_RISE, &peak_rise);
        WriteParameter("AOpeak_rise", (double)peak_rise);
```

```
GetCtrlVal(panelPara, PANELPARA2_PEAK_WIDTH, &peak_width);          peak_width *= 1.0e-6;
WriteParameter("AOpeak_width", (double)peak_width);
GetCtrlVal(panelPara, PANELPARA2_DELAY, &delay);                    delay *= 1.0e-3;
WriteParameter("AOdelay", (double)delay);

bufferSize = sampsPerCycle = (int)( (1.0/AOfrequency) / resolution );
AOrate = AOfrequency * bufferSize;

if ( (AOdata=(double*)malloc(2*bufferSize*sizeof(double)))==NULL ) {
        MessagePopup("Error","Not enough memory");
        goto Error;
}
if ( (RESETdata=(unsigned int*)malloc(bufferSize*sizeof(unsigned int)))==NULL ) {
        MessagePopup("Error","Not enough memory");
        goto Error;
}
SquareWave(bufferSize, amplitude, 1.0/sampsPerCycle, &phase, width/(1.0/AOfrequency)*100.00, AOdata);
LinEv1D(AOdata, sampsPerCycle, 1.0, amplitude, AOdata);

for (i=0; i<(peak_width/resolution); i++)
        AOdata[i] += peak_rise - rise;                  // Makes the Peak

for (i=0; i<bufferSize; i++)
        AOdata[i+bufferSize] = AOdata[i];               // for Double Beam (AO2, AO3)

// Analog Output Display
DeleteGraphPlot(panelHandle, PANEL_GRAPH, -1, VAL_DELAYED_DRAW);
SetAxisScalingMode(panelHandle, PANEL_GRAPH, VAL_LEFT_YAXIS, VAL_AUTOSCALE, min, max);
SetCtrlAttribute(panelHandle, PANEL_GRAPH, ATTR_XAXIS_GAIN, 1.0/AOrate);
SetCtrlAttribute(panelHandle, PANEL_GRAPH, ATTR_XPRECISION, (int)log10(AOrate));
PlotY(panelHandle, PANEL_GRAPH, AOdata, bufferSize, VAL_DOUBLE, VAL_FAT_LINE,
VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);

// Initilization of Photon Density Graph
DeleteGraphPlot(panelHandle, PANEL_Photon_Count, -1, VAL_IMMEDIATE_DRAW);
SetAxisScalingMode(panelHandle, PANEL_Photon_Count, VAL_LEFT_YAXIS, VAL_AUTOSCALE, 0,
        50);
SetAxisScalingMode(panelHandle, PANEL_Photon_Count, VAL_BOTTOM_XAXIS, VAL_AUTOSCALE,
        0, 50);
g_Photon_Count_Index = 0;
g_Photon_Count = (double*)malloc(sizeof(double)*PHOTON_COUNTING_TIME_SCALE);
ZeroMemory(g_Photon_Count, sizeof(g_Photon_Count));

// Initilization of Photon Delay Statistics Display
DeleteGraphPlot(panelHandle, PANEL_GRAPH, -1, VAL_DELAYED_DRAW);
SetCtrlAttribute(panelHandle, PANEL_GRAPH, ATTR_XAXIS_GAIN, 1.0);
SetCtrlAttribute(panelHandle, PANEL_GRAPH, ATTR_XPRECISION, VAL_AUTO);
SetAxisScalingMode(panelHandle, PANEL_GRAPH, VAL_LEFT_YAXIS, VAL_AUTOSCALE, 0, 150);

#ifdef PLOT_gHist_decay_time_corrected
// gTimingStep : default 555 -> 11.1 ns (90Mhz), which means each step corresponds to 20 pico second.
        GetCtrlVal(SM, PANEL_NUMERIC_TIMING_STEP, &gTimingStep);
        WriteParameter("gTimingStep", (double)gTimingStep);

        GetCtrlVal(SM, PANEL_LaserFrequency, &LaserFrequency);
        LaserFrequency *= 1.0e6;    WriteParameter("LaserFrequency", (double)LaserFrequency);
        SetAxisScalingMode(SM, PANEL_GRAPH, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0,
        1.0/LaserFrequency*1.0e9);
#else
#ifdef PLOT_gHist_laser_tick
        SetAxisScalingMode(SM, PANEL_GRAPH, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0,
        65535);
```

```
		#endif
#endif

#ifdef CONTINUOUS_TRAP
//=============================================================================
// 1. TRIGGER Signal (PFI0 (P1.0) of 6229) Configuration
//=============================================================================
		DAQmxErrChk (DAQmxCreateTask("Trigger", &gTRIGtask));
		DAQmxErrChk (DAQmxCreateDOChan(gTRIGtask, "/Dev1/port1/line0", "Trigger Pulse",
			DAQmx_Val_ChanPerLine));
		DAQmxErrChk (DAQmxStartTask(gTRIGtask));
		DAQmxErrChk (DAQmxWriteDigitalScalarU32(gTRIGtask, 1, 10.0, 0, NULL));

//=============================================================================
// 2. Continuous AO Laser Trapping Configuration (AO0, AO1)
//  - Triggered by PFI0 TRIGGER (=ao/StartTrigger)
//=============================================================================
		DAQmxErrChk (DAQmxCreateTask("Analog_Output_Laser_Trapping", &gAOtask));
		DAQmxErrChk (DAQmxCreateAOVoltageChan(gAOtask, "Dev1/ao0:1", "VoltageOut", min, max,
			DAQmx_Val_Volts, NULL));
		DAQmxErrChk (DAQmxCfgDigEdgeStartTrig(gAOtask, "/Dev1/PFI0", DAQmx_Val_Rising));
		DAQmxErrChk (DAQmxCfgSampClkTiming(gAOtask, "", AOrate, DAQmx_Val_Rising,
			DAQmx_Val_ContSamps, sampsPerCycle));
		DAQmxErrChk (DAQmxWriteAnalogF64(gAOtask, bufferSize, 0, 10.0,
			DAQmx_Val_GroupByChannel, AOdata, &written, NULL));
		// Start AO Laser Trapping (0.3 ~ 0.4 ms) with digital trigger
		DAQmxErrChk (DAQmxStartTask(gAOtask));

//=============================================================================
// 3. Continuous RESET Signal Configuration (ctr0 out : Dev1/PFI12)
//  - Triggered by PFI0 TRIGGER (=ao/StartTrigger)
//=============================================================================
		DAQmxErrChk (DAQmxCreateTask("",&gContRESETtask));
		DAQmxErrChk (DAQmxCreateCOPulseChanFreq(gContRESETtask, "Dev1/ctr0", "RESET
			Continuous Pulse", DAQmx_Val_Hz, DAQmx_Val_High, 0.0, AOfrequency,
			1.0 - (width+0.001)*AOfrequency));
		DAQmxErrChk (DAQmxCfgDigEdgeStartTrig(gContRESETtask, "/Dev1/ao/StartTrigger",
			DAQmx_Val_Rising));
		DAQmxErrChk (DAQmxCfgImplicitTiming(gContRESETtask, DAQmx_Val_ContSamps, 1000));
		DAQmxErrChk (DAQmxStartTask(gContRESETtask));	// Start Continuous RESET task
		// Should go into PFI4 in 6534 (Dev2/PFI4 : New RESET ...)
		DAQmxErrChk (DAQmxConnectTerms("/Dev1/PFI12", "/Dev2/PFI4",
			DAQmx_Val_DoNotInvertPolarity));

//=============================================================================
// 4. Continuous Droplet External TRIGGER (ctr1 out : Dev1/PFI13) with delay time
//  - Triggered by PFI0 TRIGGER (=ao/StartTrigger)
//=============================================================================
		DAQmxErrChk (DAQmxCreateTask("",&gDropletTRIGtask));
		DAQmxErrChk (DAQmxCreateCOPulseChanFreq(gDropletTRIGtask, "Dev1/ctr1", "Droplet
			External TRIGGER", DAQmx_Val_Hz, DAQmx_Val_Low, delay, AOfrequency, 0.05));
		DAQmxErrChk (DAQmxCfgDigEdgeStartTrig(gDropletTRIGtask, "/Dev1/ao/StartTrigger",
			DAQmx_Val_Rising));
		DAQmxErrChk (DAQmxCfgImplicitTiming(gDropletTRIGtask, DAQmx_Val_ContSamps,
			1000));
		// Start Droplet External TRIGGER task
		DAQmxErrChk (DAQmxStartTask(gDropletTRIGtask));
#else
#ifdef ON_DEMAND_TRAP
//=============================================================================
// We need a photo-diode signal which detects scattering from the cube. It retriggers data acquisition.
// PFI4 of 6229("/Dev1/PFI4") is necessary to be connected into.
```

```
//=============================================================================

/*//=============================================================================
// PSEUDO TRIGGER Signal (PFI0 of 6229) Configuration : TRIGchan={"/Dev1/port1/line0"}
//=============================================================================
        DAQmxErrChk (DAQmxCreateTask("Trigger", &gTRIGtask));
        DAQmxErrChk (DAQmxCreateDOChan(gTRIGtask, "Dev1/port1/line0", "Trigger Pulse",
                DAQmx_Val_ChanPerLine));
        DAQmxErrChk (DAQmxStartTask(gTRIGtask));
        DAQmxErrChk (DAQmxWriteDigitalScalarU32(gTRIGtask, 1, 10.0, 0, NULL));
        // Signal Routing for Retriggerable Items (Temporary Solution)
        DAQmxErrChk (DAQmxConnectTerms("/Dev1/PFI0", "/Dev1/RTSI0",
                DAQmx_Val_DoNotInvertPolarity));
        DAQmxErrChk (DAQmxConnectTerms("/Dev1/RTSI0", "/Dev1/PFI4",
                DAQmx_Val_DoNotInvertPolarity));*/

//=============================================================================
// 1. On Demand (Retriggerable) RESET Signal (P0.7), Initially Set HIGH
//=============================================================================
        DAQmxErrChk (DAQmxCreateTask("On Demand RESET task Init", &gRESETtask));
        DAQmxErrChk (DAQmxCreateDOChan(gRESETtask, "Dev1/port0/line7", "On Demand RESET
                channel Init", DAQmx_Val_ChanPerLine));
        DAQmxErrChk (DAQmxStartTask(gRESETtask));
        DAQmxErrChk (DAQmxWriteDigitalScalarU32(gRESETtask, 1, 10.0, 128, NULL));
        DAQmxErrChk (DAQmxStopTask(gRESETtask));
        DAQmxErrChk     (DAQmxClearTask(gRESETtask));
        // Should go into PFI4 in 6534 (Dev2/PFI4) ; Not sure it's working or not. Hardware Wiring is
        necessary (9/6/2007)
        DAQmxErrChk (DAQmxConnectTerms("/Dev1/port0/line7", "/Dev2/PFI4",
                DAQmx_Val_DoNotInvertPolarity));

//=============================================================================
// 2. Retriggerable External Timing Source for AO and RESET (ctr1 out : Dev1/PFI13, ctr1 gate : Dev1/PFI4)
//=============================================================================
        DAQmxErrChk (DAQmxCreateTask("",&gDropletTRIGtask));
        DAQmxErrChk (DAQmxCreateCOPulseChanTime(gDropletTRIGtask, "Dev1/ctr1", "AO External
                Timing Source", DAQmx_Val_Seconds, DAQmx_Val_Low, 0.0, resolution*0.5,
                resolution*0.5));
        // Triggered by SCATTERING PHOTON DETECT SIGNAL ("/Dev1/PFI4")
        DAQmxErrChk (DAQmxCfgDigEdgeStartTrig(gDropletTRIGtask, "/Dev1/PFI4",
                DAQmx_Val_Rising));
        DAQmxErrChk (DAQmxCfgImplicitTiming(gDropletTRIGtask, DAQmx_Val_FiniteSamps,
                sampsPerCycle));
        DAQmxErrChk (DAQmxSetTrigAttribute (gDropletTRIGtask, DAQmx_StartTrig_Retriggerable,
                TRUE));
        // Start Droplet External TRIGGER task
        DAQmxErrChk (DAQmxStartTask(gDropletTRIGtask));

//=============================================================================
// 3. On Demand (Retriggerable) AO Laser Trapping Configuration (AO2, AO3)
//=============================================================================
        DAQmxErrChk (DAQmxCreateTask("Analog_Output_Laser_Trapping", &gAOtask));
        DAQmxErrChk (DAQmxCreateAOVoltageChan(gAOtask, "Dev1/ao2:3", "VoltageOut", min, max,
                DAQmx_Val_Volts, NULL));
        DAQmxErrChk (DAQmxCfgSampClkTiming(gAOtask, "/Dev1/Ctr1InternalOutput", AOrate,
                DAQmx_Val_Rising, DAQmx_Val_ContSamps, sampsPerCycle));
        DAQmxErrChk (DAQmxWriteAnalogF64(gAOtask, bufferSize, 0, 10.0,
                DAQmx_Val_GroupByChannel, AOdata, &written, NULL));
        DAQmxErrChk (DAQmxStartTask(gAOtask));

//=============================================================================
// 4. On Demand (Retriggerable) RESET Signal (P0.7) Configuration
```

```
//=============================================================================
        for (i=0; i<(int)((width+0.001)/resolution); i++)
            RESETdata[i] = 0;
        for (i=(int)((width+0.001)/resolution); i<bufferSize; i++)
            RESETdata[i] = 128;
        DAQmxErrChk (DAQmxCreateTask("On Demand RESET task", &gRESETtask));
        DAQmxErrChk (DAQmxCreateDOChan(gRESETtask, "Dev1/port0/line7", "On Demand RESET
            channel", DAQmx_Val_ChanPerLine));
        DAQmxErrChk (DAQmxCfgSampClkTiming(gRESETtask, "/Dev1/Ctr1InternalOutput", AOrate,
            DAQmx_Val_Rising, DAQmx_Val_ContSamps, sampsPerCycle));
        DAQmxErrChk (DAQmxWriteDigitalU32(gRESETtask, sampsPerCycle, 0, 10.0,
            DAQmx_Val_GroupByChannel, RESETdata, &written, NULL));
        DAQmxErrChk (DAQmxStartTask(gRESETtask));
    #endif
    #endif

//=============================================================================
// 5. Data Acquisition (Continuous Pattern I/O; sampling is controlled by REQ signal at /Dev2/PFI2)
//=============================================================================
    GetCtrlVal(panelHandle, PANEL_DIPORTSPHYSICAL, chan);
    WriteCharParameter("DI_Ports_Physical", chan);
    GetCtrlVal(panelHandle, PANEL_SAMPSTOREAD, &sampsToRead);
    WriteParameter("sampsToRead", (double)sampsToRead);
    GetCtrlVal(panelHandle, PANEL_RATE, &rate);
    WriteParameter("rate", (double)rate);
    DAQmxErrChk (DAQmxCreateTask("Digital_Input_Data_Acquisition", &gDItask));
    DAQmxErrChk (DAQmxCreateDIChan(gDItask, chan, "DI Pattern IO", DAQmx_Val_ChanForAllLines));
    DAQmxErrChk (DAQmxCfgSampClkTiming(gDItask, REQclockSource6534, rate, DAQmx_Val_Rising,
        DAQmx_Val_ContSamps, (10*sampsToRead<10000) ? 10000 : 10*sampsToRead));
    DAQmxErrChk (DAQmxStartTask(gDItask));      // Start Data Acquisition ( ~ 140 ms)
    gTrapDAQRunning = 1;

//=============================================================================
// 6. Change Detection Configuration for DAQmxRead / Write calling
// (P0.0) is externally (by wire) connected from
// RESET (Dev1/PFI12 at CONTINUOUS_TRAP, (P0.7) at ON_DEMAND_TRAP).
// It detects the change (the rising edge) of RESET, calling DATA acqusition / HDD Writing function.
//=============================================================================
    DAQmxErrChk (DAQmxCreateTask("Change Detection", &gChangeDetectiontask));
    DAQmxErrChk (DAQmxCreateDIChan(gChangeDetectiontask, "Dev1/port0/line0", "Change Detection",
        DAQmx_Val_ChanPerLine));
    DAQmxErrChk (DAQmxCfgChangeDetectionTiming(gChangeDetectiontask, "Dev1/port0/line0", NULL,
        DAQmx_Val_ContSamps, 1000));
    DAQmxErrChk (DAQmxRegisterSignalEvent(gChangeDetectiontask, DAQmx_Val_ChangeDetectionEvent,
        0, ChangeDetectionCallback, NULL));
    DAQmxErrChk (DAQmxStartTask(gChangeDetectiontask));

    if ( (gData = (unsigned int*)malloc(sampsToRead*sizeof(unsigned int)))==NULL ) {
        MessagePopup("Error","Not enough memory for data for gData");
        goto Error;
    }
    if ( (gDataValues = (unsigned int*)malloc(sampsToRead*sizeof(unsigned int)))==NULL ) {
        MessagePopup("Error","Not enough memory for data for gDataValues");
        goto Error;
    }

//=============================================================================
// 7. Create Thread-Safe Queues to transfer data between threads
//=============================================================================
#ifdef THREAD_SAFETY_QUE
    if (CmtNewTSQ (10*sampsToRead, sizeof(double), OPT_TSQ_DYNAMIC_SIZE, &g_timerQueueHdl)
    < 0) {
```

```
                DAQmxError = -1;
                goto Error;
        }
        if (CmtNewTSQ (10*sampsToRead, sizeof(unsigned int), OPT_TSQ_DYNAMIC_SIZE, &g_dataQueueHdl)
        < 0) {
                DAQmxError = -2;
                goto Error;
        }
#endif

//==============================================================================
// 8. Initialize Thread-Safe Variables
//==============================================================================
        InitializeIndex();
        InitializeReadIndex();
        InitializeWriteIndex();
        InitializeAutoCalibrationIndex();
        SetIndex(0);
        SetReadIndex(0);
        SetWriteIndex(0);
        SetAutoCalibrationIndex(1);

//==============================================================================
// 8.1. Write Log File
//==============================================================================
QueryPerformanceCounter(&gTime2);
        time12 = (double)(gTime2.QuadPart - gTime1.QuadPart) / (double)gTicksPerSecond.QuadPart * 1000.0;
        sprintf(errBuff, "%.06f ms", time12);
        SetCtrlVal(SM, PANEL_HEXVALUE, errBuff);  // 1 : sm.uir

#ifdef WRITE_LOG_FILE
        fprintf(gFp, "[%s, %s] Data Collection Starts\n", DateStr(), TimeStr());
        fprintf(gFp_thread, "[%s, %s] Initial Time Delay from AO Trapping : %.06f ms\n", DateStr(), TimeStr(),
        time12);
#endif

//==============================================================================
// 9. Auto-calibration
//==============================================================================
        // Only if Autocalibration is checked "Yes"...
        if (bIsAutoCalibration_Checked) {
                LARGE_INTEGER timer1, timer2;
                int index=1;
                char temp[256];

QueryPerformanceCounter(&timer1);
                GetCtrlVal(AUTO_CALIBRATION, PANEL_CALI_NUMERIC_CALI_PERIOD,
                        &g_AutoCalibrationDelay);               g_AutoCalibrationDelay *= 60.0;

                // 9.1. Create the new timer for Auto-Calibration Pulse ; It takes g_AutoCalibrationDelay before
                launching the very first thread
                g_AutoCalibrationTimerId = NewAsyncTimer (g_AutoCalibrationDelay, -1, 0,
                        AutoCalibration_Callback, NULL);
                if (g_AutoCalibrationTimerId <= 0) {
                        sprintf(errBuff, "Async update timer could not be created due to the error of %d",
                                g_AutoCalibrationTimerId);
                        MessagePopup("Async Timer", errBuff);
                        WriteLog(errBuff, FALSE);
                        g_AutoCalibrationTimerId = 0;
                        DAQmxError = -4;
                        goto Error;
                }
```

```
                    // 9.2. UV Laser Shutter Controlling Pulse (P0.16) of 6229) Configuration
                    DAQmxErrChk (DAQmxCreateTask("Autocalibration Test Pulse", &gAutoCalibration_task));
                    DAQmxErrChk (DAQmxCreateDOChan(gAutoCalibration_task, "/Dev1/port0/line16",
                            "Autocalibration Test Pulse", DAQmx_Val_ChanPerLine));
                    DAQmxErrChk (DAQmxStartTask(gAutoCalibration_task));
                    DAQmxErrChk (DAQmxWriteDigitalScalarU32(gAutoCalibration_task, 1, 10.0, 0, NULL));

                    // 9.3. the First Calibration
                    StopTASKs(index);
                    Calibration_Update(index);
                    StartTASKs(index);
QueryPerformanceCounter(&timer2);

                    // Log Writing
                    time12 = (double)(timer2.QuadPart - timer1.QuadPart) / (double)gTicksPerSecond.QuadPart;
                    sprintf(errBuff, "[%s, %s] Calibration [#%d] (W%d) : %.06f sec", DateStr(), TimeStr(), index,
                            GetWriteIndex(), time12);
                    SetCtrlVal(SM, PANEL_HEXVALUE, errBuff);
                    #ifdef WRITE_LOG_FILE
                            fprintf(gFp_thread, "%s\n", errBuff);
                    #endif
            }

Error:
            if ( DAQmxFailed(DAQmxError) ) {
                    DAQmxGetExtendedErrorInfo(errBuff, 2048);
                    MessagePopup("DAQmx Error", errBuff);
                    WriteLog(errBuff, FALSE);
            }
            if ( AOdata )
                    free(AOdata);
            if ( RESETdata )
                    free(RESETdata);
            return DAQmxError;
}


//==============================================================================
// Change Detection - Detect the Rising Edge of RESET signal
//==============================================================================
int CVICALLBACK ChangeDetectionCallback(TaskHandle taskHandle, int32 signalID, void *callbackData) {

            // Launch a DATA Acquisition (Reading) Thread
            CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE, DAQ_thread, NULL,
                    &gDAQThreadId);
            return 0;
}
```

C.2.4 Data reading thread (sm_fn.c)

```
//==============================================================================
// DATA acquisition (Reading) Thread
//==============================================================================
int CVICALLBACK DAQ_thread (void *functionData) {
            double currentTime = 0.0;
            double deltaTime   = 0.0;
```

```
	char buff[100];
	unsigned int sampsToRead;
	int	index=0, sampsRead=0, totsampsRead=0, i;
	LARGE_INTEGER timer1, timer2, timer3, timer4, timer5, ticksPerSecond, timerC;
	double time12, time23, time34, timeC5;
	double timeValue[1];
	DWORD dwBytesWritten=0;
	DWORD NumberOfBytesTransferred=0;
	double AOfrequency;

	EnterCriticalSection( &gLock_READ );
QueryPerformanceCounter(&timer1);
QueryPerformanceCounter(&timer3);
	index = GetReadIndex();
	SetReadIndex(++index);

	#ifdef PSEUDO_SAMPLE // for testing purpose
		for (i=0; i<ksj; i++) {
			gData[i] = (i % 1024) | 32768;
			//gData[i] = (rand() % 1024) | 32768;
		}
		sampsRead = ksj;
	#else
		GetCtrlVal(SM, PANEL_SAMPSTOREAD, &sampsToRead);
		DAQmxReadDigitalU32(gDItask, DAQmx_Val_Auto, 10.0, DAQmx_Val_GroupByChannel,
			gData, sampsToRead, &sampsRead, NULL);	//0.3 ~ 0.5 ms for 300 samples
	#endif

	//if (sampsRead > 0) {
	#ifdef THREAD_SAFETY_QUE	// Current Version
		// Queue Writing
		CmtWriteTSQData (g_dataQueueHdl, gData, sampsRead, TSQ_INFINITE_TIMEOUT,
			NULL);	// ~ 2ms for 10^5 samples
	#endif

	#ifndef NEW_DATA_STORAGE_THREAD_AT_CHANGE_DETECTION
		#ifdef NEW_DATA_STORAGE_THREAD_AFTER_DAQ		// Current Version
			gnItemsRead = sampsRead;

			// Launch a DATA Storage (Writing) Thread
			CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
				Storage_thread, NULL, &gDataStorageThreadId2);
		#endif
	#endif
	//}

QueryPerformanceCounter(&timer4);
	timeValue[0] = (double)(timer1.QuadPart - gTime1.QuadPart) / (double)gTicksPerSecond.QuadPart *
		1000.0;

	#ifdef WRITE_LOG_FILE
		time34 = (double)(timer4.QuadPart - timer3.QuadPart) /
			(double)gTicksPerSecond.QuadPart * 1000.0;
		fprintf(gFp_thread, "R\t%d\t%.06f\tR%d\t%.06f\n", index, timeValue[0], sampsRead,
			time34);
	#endif

	gTime1.QuadPart = timer1.QuadPart;

	#ifdef DISPLAY_READING_LOG
		GetCtrlVal(PARAMETER2, PANELPARA2_FREQUENCY, &AOfrequency);
QueryPerformanceCounter(&timer2);
```

```
                time12 = (double)(timer2.QuadPart - timer1.QuadPart) / (double)gTicksPerSecond.QuadPart *
                        1000.0;
                if (index % (int)AOfrequency == 1) {
                        sprintf(buff, "R(%d) %.06f ms %d : %d-%d", index, time12, sampsRead,
                        gData[max(0,totsampsRead-1)]&64512, gData[max(0,totsampsRead-1)]&1023);
                        SetCtrlVal(SM, PANEL_HEXVALUE, buff);       // 1 : sm.uir
                }
        #endif

        LeaveCriticalSection( &gLock_READ );
        return 0;
}
```

C.2.5 Massive data storage thread (sm_fn.c)

```
//==============================================================================
// Massive Data Storage (HDD Writing) Thread
//==============================================================================
int CVICALLBACK Storage_thread(void *functionData) {
        int i=0, pmt, index=0;
        char buff[1000]={'\0'};
        LARGE_INTEGER timer1, timer2, timer3, timer4;
        double time12, time34;
        DWORD dwBytesWritten=0;
        DWORD NumberOfBytesTransferred=0;
        int nItemsData=0;
        int temp=0;
        double AOfrequency, width;
        double *graph_X=NULL;

        EnterCriticalSection( &gLock_WRITE );
QueryPerformanceCounter(&timer1);
        index = GetWriteIndex();
        SetWriteIndex(++index);

#ifdef THREAD_SAFETY_QUE        // Current Version
        // Getting data from Queue
        gnItemsRead = 0;
        CmtGetTSQAttribute (g_dataQueueHdl, ATTR_TSQ_ITEMS_IN_QUEUE, &nItemsData);
        if (nItemsData > 0) {
                if ((gDataValues = (unsigned int*) realloc (gDataValues, nItemsData*sizeof(unsigned int))) !=
                NULL) {
                        gnItemsRead = CmtReadTSQData (g_dataQueueHdl, gDataValues, nItemsData,
                                TSQ_INFINITE_TIMEOUT, 0);
                        if (gnItemsRead > 0) {
                                gnItemsTotalRead += gnItemsRead;
                        #ifdef OVERLAPPED_IO    // Current Version
                                if (GetOverlappedResult (ghFile, &gOverlapped, &NumberOfBytesTransferred,
                                TRUE) == FALSE) {
                                        sprintf (buff, "GetOverlappedResult %dth : %d bytes transfer
                                                (ERROR %d)\n", GetWriteIndex(),
                                                NumberOfBytesTransferred,  GetLastError());
                                        MessagePopup("GetOverlappedResult Error", buff);
                                        WriteLog(buff, FALSE);
                                }
                                gFileSize.QuadPart += NumberOfBytesTransferred;
                                gOverlapped.Offset     = gFileSize.LowPart;
```

```
                                        gOverlapped.OffsetHigh = gFileSize.HighPart;
                                        WriteFile( ghFile, gDataValues, gnItemsRead*sizeof(unsigned int),
                                                &dwBytesWritten, &gOverlapped );
                                #else
                                        WriteFile( ghFile, gDataValues, gnItemsRead*sizeof(unsigned int),
                                                &dwBytesWritten, NULL );
                                        NumberOfBytesTransferred = dwBytesWritten;
                                #endif
                                }
                        }
                }
#else
#ifdef OVERLAPPED_IO
                if (GetOverlappedResult (ghFile, &gOverlapped, &NumberOfBytesTransferred, TRUE) == FALSE) {
                        sprintf (buff, "GetOverlappedResult %dth : %d bytes transfer (ERROR %d)\n", GetWriteIndex(),
                                NumberOfBytesTransferred, GetLastError());
                        MessagePopup("GetOverlappedResult Error", buff);
                        WriteLog(buff, FALSE);
                }
                gFileSize.QuadPart += NumberOfBytesTransferred;
                gOverlapped.Offset     = gFileSize.LowPart;
                gOverlapped.OffsetHigh = gFileSize.HighPart;
                WriteFile( ghFile, gData, gnItemsRead*sizeof(unsigned int), &dwBytesWritten, &gOverlapped );
#else
                WriteFile( ghFile, gData, gnItemsRead*sizeof(unsigned int), &dwBytesWritten, NULL );
                NumberOfBytesTransferred = dwBytesWritten;
#endif
#endif

QueryPerformanceCounter(&timer2);
                time12 = (double)(timer2.QuadPart - timer1.QuadPart) / (double)gTicksPerSecond.QuadPart * 1000.0;

#ifdef DISPLAY_WRITING_LOG
                GetCtrlVal(PARAMETER2, PANELPARA2_WIDTH, &width);       width *= 1.0e-6;
                GetCtrlVal(PARAMETER2, PANELPARA2_FREQUENCY, &AOfrequency);

                // every 1.0 sec, it displays histogram...
                if (index % (int)AOfrequency == (int)(AOfrequency*0.5) + 1) {
QueryPerformanceCounter(&timer3);
                        fprintf(gFp, "%g\n", gnItemsTotalRead / ( (width+0.001)*AOfrequency ));

                        if (GetgRTPhotonCount() == 1) {
                                //Take data for photon density plot here (Krish)
                                if (g_Photon_Count_Index < PHOTON_COUNTING_TIME_SCALE) {
                                        g_Photon_Count[g_Photon_Count_Index] = gnItemsTotalRead /
                                                ( (width+0.001)*AOfrequency );
                                } else {
                                        int j;
                                        for (j=1; j<PHOTON_COUNTING_TIME_SCALE; j++)
                                                g_Photon_Count[j-1] = g_Photon_Count[j];
                                        g_Photon_Count[PHOTON_COUNTING_TIME_SCALE-1] =
                                                gnItemsTotalRead / ( (width+0.001)*AOfrequency );
                                }
                                g_Photon_Count_Index++;

                                DeleteGraphPlot(SM, PANEL_Photon_Count, -1, VAL_DELAYED_DRAW);
                                if (g_Photon_Count_Index < PHOTON_COUNTING_TIME_SCALE)
                                        PlotY (SM, PANEL_Photon_Count, g_Photon_Count, g_Photon_Count_Index,
                                                VAL_DOUBLE, VAL_FAT_LINE, VAL_EMPTY_SQUARE,
                                                VAL_SOLID, 1, VAL_YELLOW);
                                else
                                        PlotY (SM, PANEL_Photon_Count, g_Photon_Count,
```

```
                    PHOTON_COUNTING_TIME_SCALE, VAL_DOUBLE,
                    VAL_FAT_LINE, VAL_EMPTY_SQUARE, VAL_SOLID, 1,
                    VAL_YELLOW);
}
gnItemsTotalRead = 0;

if (gnItemsRead > 0) {
#ifdef PLOT_gHist_decay_time_corrected
        double LaserFrequency;
        unsigned int data;

        //Init_Histrograms();
        graph_X = (double*) malloc(gTimingStep * sizeof(double));
        ZeroMemory(graph_X, sizeof(graph_X));

        for (i=0; i<gnItemsRead; i++) {
        #ifdef THREAD_SAFETY_QUE
                data = gDataValues[i];
        #else
                data = gData[i];
        #endif
                // Low 10 bits for Laser time gap
                switch ( data & 64512 ) {
                        case 1024 : gHist_decay_time1[data & 1023]++;
                                break;
                        case 2048 : gHist_decay_time2[data & 1023]++;
                                break;
                        case 4096 : gHist_decay_time3[data & 1023]++;
                                break;
                        case 8192 : gHist_decay_time4[data & 1023]++;
                                break;
                        case 16384 : gHist_decay_time5[data & 1023]++;
                                break;
                        case 32768 : gHist_decay_time6[data & 1023]++;
                                break;
                        default : break;
                }
                //gHist_decay_time[data & 1023]++;
        }
        // gTimingStep : default 555 -> 11.1 ns (90Mhz), which means each step
        corresponds to 20 pico second.
        GetCtrlVal(SM, PANEL_NUMERIC_TIMING_STEP, &gTimingStep);
        // Data Correction
       correct_with_offset(gHist_decay_time1, gHist_decay_time_corrected1, 0, 1024,
                gTimingStep, gCorrections1, gPMT1_Offset);
       correct_with_offset(gHist_decay_time2, gHist_decay_time_corrected2, 0, 1024,
                gTimingStep, gCorrections2, gPMT2_Offset);
       correct_with_offset(gHist_decay_time3, gHist_decay_time_corrected3, 0, 1024,
                gTimingStep, gCorrections3, gPMT3_Offset);
       correct_with_offset(gHist_decay_time4, gHist_decay_time_corrected4, 0, 1024,
                gTimingStep, gCorrections4, gPMT4_Offset);
       correct_with_offset(gHist_decay_time5, gHist_decay_time_corrected5, 0, 1024,
                gTimingStep, gCorrections5, gPMT5_Offset);
       correct_with_offset(gHist_decay_time6, gHist_decay_time_corrected6, 0, 1024,
                gTimingStep, gCorrections6, gPMT6_Offset);

        GetCtrlVal(SM, PANEL_LaserFrequency, &LaserFrequency);
        LaserFrequency *= 1.0e6;

        for (i=0; i<gTimingStep; i++) {
                gHist_decay_time_corrected[i] = gHist_decay_time_corrected1[i] +
                gHist_decay_time_corrected2[i] + gHist_decay_time_corrected3[i]
```

```
                                        + gHist_decay_time_corrected4[i] + gHist_decay_time_corrected5[i]
                                        + gHist_decay_time_corrected6[i];
                                        //+ gHist_decay_time_corrected0[i];
                                        graph_X[i] = 1.0/LaserFrequency*1.0e9*i/gTimingStep;
                                }

                                DeleteGraphPlot(SM, PANEL_GRAPH, -1, VAL_DELAYED_DRAW);
                                PlotXY(SM, PANEL_GRAPH, graph_X, gHist_decay_time_corrected,
                                        gTimingStep, VAL_DOUBLE, VAL_DOUBLE, VAL_THIN_LINE,
                                        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
                        #else
                        #ifdef PLOT_gHist_laser_tick
                                ZeroMemory(gHist_laser_tick, sizeof(gHist_laser_tick));
                                for (i=0; i<gnItemsRead; i++) {
                                // High 16 bits for actual time gap
                                #ifdef THREAD_SAFETY_QUE
                                        temp = (gDataValues[i] & 4294901760) >> 16;
                                        gHist_laser_tick[temp]++;
                                #else
                                        gHist_laser_tick[(gData[i] & 4294901760) >> 16]++;
                                #endif
                                }
                                DeleteGraphPlot(SM, PANEL_GRAPH, -1, VAL_DELAYED_DRAW);
                                PlotY(SM, PANEL_GRAPH, gHist_laser_tick, 65536,
                                        VAL_UNSIGNED_INTEGER, VAL_THIN_LINE,
                                        VAL_EMPTY_SQUARE, VAL_SOLID, 1, VAL_RED);
                        #endif
                        #endif
                        }
QueryPerformanceCounter(&timer4);
                        time34 = (double)(timer4.QuadPart - timer3.QuadPart) /
                                (double)gTicksPerSecond.QuadPart * 1000.0;
                        sprintf(buff, "W(%d) %.06f ms (Writing) %.06f ms (Histogram) W%d %d/%d\n", index,
                        time12, time34, gnItemsRead, NumberOfBytesTransferred, sizeof(unsigned int));
                        SetCtrlVal (SM, PANEL_HEXVALUE, buff);
                }
        #endif

QueryPerformanceCounter(&timer2);
        time12 = (double)(timer2.QuadPart - timer1.QuadPart) / (double)gTicksPerSecond.QuadPart * 1000.0;

#ifdef WRITE_LOG_FILE
        fprintf(gFp_thread, "W\t%d\t%.06f\tW%d\t%d/%d\n", index, time12, gnItemsRead,
                NumberOfBytesTransferred, sizeof(unsigned int));
#endif
        if (graph_X)
                free (graph_X);

        LeaveCriticalSection( &gLock_WRITE );
        return 0;
}
```

C.2.6 Analysis tool (sm.c in SM_UNIX version)

```
//==============================================================================
// Main Function
//==============================================================================
```

```
int main (int argc, char *argv[]) {
	time_t s1, s2;
	int time_diff, mode;

	s1 = time (NULL);
	if (argc >= 2)
		mode = ReadINIparameters(argv[1]);
	else
		mode = ReadINIparameters("sm.ini");		// Default
	get_gPMT_Offsets();

	// Time Calibration for The BOX : Default 555 steps during 11.1 ns (90Mhz pulse laser), which means each
	step corresponds to 20 pico second.
	TimeCalibration();

	switch (mode) {
		case 1 :	Analyze_UNIX();			// Analysis of raw data file
				break;
		case 2 :	Histogram_for_TimeCalibration();		// histogram for time calibration
				break;
		case 3 :	PhotonCountAnalyze_UNIX();			// Photon Count Analysis
				break;
		case 4 :	Selective_PhotonCountAnalyze_UNIX();	// Selective Photon Count Analysis
				break;
		case 5 :	Concatenate_PhotonCountAnalysis_UNIX();		// Photon Count Analysis
				break;
		default :
				break;
	}
	s2 = time (NULL);
	time_diff = (int)(s2-s1);
	printf("\n> Analysis finished at [%s]", DateTimeToString());
	printf("\n> Execution Time = %d seconds (= %g minutes)\n\n", time_diff, time_diff/60.0);
	return 0;
}


//=============================================================================================
// Analyze function - UNIX Version
//=============================================================================================
int Analyze_UNIX(void) {
	FILE *hFile;
	int bIsBINARY = 1;
	unsigned int i;
	int BufferSize = 8*1024;			// 8 * 1024 Samples
	double *graph_X=NULL, coeff;
	unsigned int nBytesToRead;
	unsigned int nItemsToRead;
	unsigned int nItemsRead;
	unsigned int *inBuffer;
	unsigned int index=0;
	char buff[255], buff2[255];
	FILE *fp, *fp1, *fp2, *fp3, *fp4, *fp5, *fp6, *fp0, *fp_bad, *fp_out_of_range;
	FILE *fpHist, *fpRawHist;
	int mode1, mode2, mode3, mode4, mode5, mode6;
	unsigned int raw_decay_start1, raw_decay_end1, raw_decay_start2, raw_decay_end2, raw_decay_start3,
		raw_decay_end3;
	unsigned int raw_decay_start4, raw_decay_end4, raw_decay_start5, raw_decay_end5, raw_decay_start6,
		raw_decay_end6;
	double direct_corrected_decay, laser_tick=0.0, DecayStart=0.0, DecayEnd=0.0;
	unsigned int datapoint=0, PMTBuffer=0, willbeRemoved, raw_decay;
	unsigned int unwrapping_index=0, bad_count=0, error_count=0, out_of_range_count=0;
```

```
nBytesToRead = sizeof(unsigned int) * BufferSize;		// 32 KB block unit & unsigned int : 4 Bytes
nItemsToRead = BufferSize;
inBuffer = (unsigned int *)malloc(nBytesToRead);
graph_X = (double*) malloc(g_TimingStep * sizeof(double));

// determine the file extension (binary? or ascii?)
if ( strstr(g_RawDataFile, ".dat") != NULL )
	bIsBINARY = 1;
else if ( strstr(g_RawDataFile, ".txt") != NULL )
	bIsBINARY = 0;

// Opens an empty file for writing. If the given file exists, its contents are destroyed
fp = fopen(g_AnalyzedFile, "w");
fprintf(fp, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_HIST.txt");
fpHist = fopen(buff, "w");

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_RAW_HIST.txt");
fpRawHist = fopen(buff, "w");

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_PMT1.txt");
fp1 = fopen(buff, "w");
fprintf(fp1, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_PMT2.txt");
fp2 = fopen(buff, "w");
fprintf(fp2, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_PMT3.txt");
fp3 = fopen(buff, "w");
fprintf(fp3, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_PMT4.txt");
fp4 = fopen(buff, "w");
fprintf(fp4, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_PMT5.txt");
fp5 = fopen(buff, "w");
fprintf(fp5, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_PMT6.txt");
fp6 = fopen(buff, "w");
fprintf(fp6, "[%s]\n", DateTimeToString());
```

```
strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_PMT0.txt");
fp0 = fopen(buff, "w");
fprintf(fp0, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
strcat(buff, "_out_of_range.txt");
fp_out_of_range = fopen(buff, "w");
fprintf(fp_out_of_range, "[%s]\n", DateTimeToString());

strncpy(buff, g_AnalyzedFile, strlen(g_AnalyzedFile)-4);
buff[strlen(g_AnalyzedFile)-4] = '\0';
sprintf(buff2, "_bad%02d.txt", g_PhotonRemoval);
strcat(buff, buff2);
fp_bad = fopen(buff, "w");
fprintf(fp_bad, "[%s]\n", DateTimeToString());

DecayStart = g_DecayWinStart / (1.0/g_LaserFrequency) * g_TimingStep;
DecayEnd = g_DecayWinEnd / (1.0/g_LaserFrequency) * g_TimingStep;

mode1 = calc_raw_decay(&raw_decay_start1, &raw_decay_end1, DecayStart, DecayEnd, gCorrections1,
        gPMT1_Offset, (int)g_TimingStep);
mode2 = calc_raw_decay(&raw_decay_start2, &raw_decay_end2, DecayStart, DecayEnd, gCorrections2,
        gPMT2_Offset, (int)g_TimingStep);
mode3 = calc_raw_decay(&raw_decay_start3, &raw_decay_end3, DecayStart, DecayEnd, gCorrections3,
        gPMT3_Offset, (int)g_TimingStep);
mode4 = calc_raw_decay(&raw_decay_start4, &raw_decay_end4, DecayStart, DecayEnd, gCorrections4,
        gPMT4_Offset, (int)g_TimingStep);
mode5 = calc_raw_decay(&raw_decay_start5, &raw_decay_end5, DecayStart, DecayEnd, gCorrections5,
        gPMT5_Offset, (int)g_TimingStep);
mode6 = calc_raw_decay(&raw_decay_start6, &raw_decay_end6, DecayStart, DecayEnd, gCorrections6,
        gPMT6_Offset, (int)g_TimingStep);

// Analysis of Raw BINARY Data
if (bIsBINARY) {
        int old_laser_tick=0, new_laser_tick=0;

        hFile = fopen(g_RawDataFile, "rb");
        if (hFile == NULL)
                return -1;

        while( !feof( hFile ) ) {                       // Check for end of file.
                nItemsRead = fread(inBuffer, sizeof(unsigned int), nItemsToRead, hFile);
                if( ferror( hFile ) ) {
                        printf( "Read error\n" );
                        break;
                }
                index++;

                for (i=0; i<nItemsRead; i++) {          // 8 * 1024 Samples for maximum reading
                        new_laser_tick = (inBuffer[i] & 4294901760) >> 16;
                        raw_decay = inBuffer[i] & 1023;

                        if (new_laser_tick < old_laser_tick) {
                                old_laser_tick -= 65536;
                                if (g_isUnwrapping)
                                        unwrapping_index++;
                        }
                        laser_tick = (double)new_laser_tick + 65536.0*(double)unwrapping_index;
```

```
// Incorporate removal of photons spaced by g_PhotonRemoval laser
pulses/ticks or less (Modified by SJKim, at Jan 4. 2008)
willbeRemoved = (new_laser_tick-old_laser_tick <= g_PhotonRemoval)? 1 : 0;
old_laser_tick = new_laser_tick;

switch ( inBuffer[i] & 64512 ) {
        case 1024 :     PMTBuffer = 1;
                if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                &direct_corrected_decay, raw_decay_start1,
                raw_decay_end1, gDirectCorrection1, gPMT1_Offset,
                g_TimingStep,willbeRemoved, fp_bad, &bad_count,
                mode1, fp_out_of_range, &out_of_range_count, fp1,
                gHist_decay_time1) < 0 )
                        continue;
                        break;
        case 2048 :     PMTBuffer = 2;
                if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                &direct_corrected_decay, raw_decay_start2,
                raw_decay_end2, gDirectCorrection2, gPMT2_Offset,
                g_TimingStep, willbeRemoved, fp_bad, &bad_count,
                mode2, fp_out_of_range, &out_of_range_count, fp2,
                gHist_decay_time2) < 0 )
                        continue;
                        break;
        case 4096 :     PMTBuffer = 3;
                if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                &direct_corrected_decay, raw_decay_start3,
                raw_decay_end3, gDirectCorrection3, gPMT3_Offset,
                g_TimingStep, willbeRemoved, fp_bad, &bad_count,
                mode3, fp_out_of_range, &out_of_range_count, fp3,
                gHist_decay_time3) < 0 )
                        continue;
                        break;
        case 8192 :     PMTBuffer = 4;
                if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                &direct_corrected_decay, raw_decay_start4,
                raw_decay_end4, gDirectCorrection4, gPMT4_Offset,
                g_TimingStep, willbeRemoved, fp_bad, &bad_count,
                mode4, fp_out_of_range, &out_of_range_count, fp4,
                gHist_decay_time4) < 0 )
                        continue;
                        break;
        case 16384:     PMTBuffer = 5;
                if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                &direct_corrected_decay, raw_decay_start5,
                raw_decay_end5, gDirectCorrection5, gPMT5_Offset,
                g_TimingStep, willbeRemoved, fp_bad, &bad_count,
                mode5, fp_out_of_range, &out_of_range_count, fp5,
                gHist_decay_time5) < 0 )
                        continue;
                        break;
        case 32768:     PMTBuffer = 6;
                if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                &direct_corrected_decay, raw_decay_start6,
                raw_decay_end6, gDirectCorrection6, gPMT6_Offset,
                g_TimingStep, willbeRemoved, fp_bad, &bad_count,
                mode6, fp_out_of_range, &out_of_range_count, fp6,
                gHist_decay_time6) < 0 )
                        continue;
                        break;
        default :   PMTBuffer = (inBuffer[i] & 64512);
                // ERROR - Krish(12/30/07): Instead of showing PMT# as
```

```
                                                // zero it will show the actual 6 digit binary number
                                                if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                                &direct_corrected_decay, raw_decay_start1,
                                                raw_decay_end1, gDirectCorrection1, gPMT1_Offset,
                                                g_TimingStep, willbeRemoved, fp_bad, &bad_count,
                                                mode1, fp_out_of_range, &out_of_range_count, fp0,
                                                gHist_decay_time0) < 0 )
                                                        continue;
                                                        error_count++;
                                        }
                                        fprintf(fp, "%.15g\t%u\t%04u\t%04f\n", laser_tick, PMTBuffer, raw_decay,
                                                direct_corrected_decay);
                                        if (g_isRawHistrogram)
                                                gHist_decay_time[raw_decay]++; // Low 10 bits for Laser time gap
                                }
                                datapoint += i;
                        }
                        printf("Binary -> ASCII\n");
                        printf("Total Number of Photons, within the range (%g ns ~ %g ns) = %d\n",
                        g_DecayWinStart*1.0e9, g_DecayWinEnd*1.0e9, datapoint - bad_count - out_of_range_count);
                        fclose(hFile);
}
// Analysis of Already-Analyzed ASCII Data Analyzing
else {
                        int err;
                        FILE *fpTxt;
                        double decay_corrected=0.0, new_laser_tick=0.0, old_laser_tick=0.0;
                        char temp[50];

                        fpTxt = fopen(g_RawDataFile, "r");
                        if (fpTxt == NULL)
                                return -1;

                        sprintf(temp, "[%s]\n", DateTimeToString());
                        fseek(fpTxt, strlen(temp), SEEK_SET);

                        while ( err = fscanf (fpTxt, "%lf\t%d\t%d\t%lf", &new_laser_tick, &PMTBuffer, &raw_decay,
                        &decay_corrected) > 0 ) {
                                index++;

                                if (new_laser_tick < old_laser_tick) {
                                        old_laser_tick -= 65536.0;
                                        if (g_isUnwrapping)
                                                unwrapping_index++;
                                }
                                laser_tick = new_laser_tick + 65536.0*(double)unwrapping_index;

                                // Incorporate removal of photons spaced by g_PhotonRemoval laser pulses/ticks or less
                                // (Modified by SJKim, at Jan 4. 2008)
                                willbeRemoved = (new_laser_tick-old_laser_tick <= g_PhotonRemoval) ? 1 : 0;
                                old_laser_tick = new_laser_tick;

                                switch ( PMTBuffer ) {
                                        case 1 :    if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                                    &direct_corrected_decay, raw_decay_start1, raw_decay_end1,
                                                    gDirectCorrection1, gPMT1_Offset, g_TimingSte, willbeRemoved,
                                                    fp_bad, &bad_count,  mode1, fp_out_of_range,&out_of_range_count,
                                                    fp1, gHist_decay_time1) < 0 )
                                                            continue;
                                                    break;
                                        case 2 :    if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                                    &direct_corrected_decay, raw_decay_start2, raw_decay_end2,
```

```
                                        gDirectCorrection2, gPMT2_Offset, g_TimingStep, willbeRemoved,
                                        fp_bad, &bad_count,  mode2, fp_out_of_range,&out_of_range_count,
                                        fp2, gHist_decay_time2) < 0 )
                                                continue;
                                        break;
                                case 3 :    if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                        &direct_corrected_decay, raw_decay_start3, raw_decay_end3,
                                        gDirectCorrection3, gPMT3_Offset, g_TimingStep, willbeRemoved,
                                        fp_bad, &bad_count, mode3, fp_out_of_range, &out_of_range_count,
                                        fp3, gHist_decay_time3) < 0 )
                                                continue;
                                        break;
                                case 4 :    if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                        &direct_corrected_decay, raw_decay_start4, raw_decay_end4,
                                        gDirectCorrection4, gPMT4_Offset, g_TimingStep, willbeRemoved,
                                        fp_bad, &bad_count, mode4, fp_out_of_range, &out_of_range_count,
                                        fp4, gHist_decay_time4) < 0 )
                                                continue;
                                        break;
                                case 5 :    if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                        &direct_corrected_decay, raw_decay_start5, raw_decay_end5,
                                        gDirectCorrection5, gPMT5_Offset, g_TimingStep, willbeRemoved,
                                        fp_bad, &bad_count, mode5, fp_out_of_range, &out_of_range_count,
                                        fp5, gHist_decay_time5) < 0 )
                                                continue;
                                        break;
                                case 6 :    if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                        &direct_corrected_decay, raw_decay_start6, raw_decay_end6,
                                        gDirectCorrection6, gPMT6_Offset, g_TimingStep, willbeRemoved,
                                        fp_bad, &bad_count, mode6, fp_out_of_range, &out_of_range_count,
                                        fp6, gHist_decay_time6) < 0 )
                                                continue;
                                        break;
                                default :   if ( calc_hist(laser_tick, PMTBuffer, raw_decay,
                                        &direct_corrected_decay, raw_decay_start1, raw_decay_end1,
                                        gDirectCorrection1, gPMT1_Offset, g_TimingStep, willbeRemoved,
                                        fp_bad, &bad_count, mode1, fp_out_of_range, &out_of_range_count,
                                        fp0, gHist_decay_time0) < 0 )
                                                continue;
                                        error_count++;
                        }
                        fprintf(fp, "%.15g\t%u\t%04u\t%04f\n", laser_tick, PMTBuffer, raw_decay,
                                direct_corrected_decay);
                        if (g_isRawHistrogram)
                                gHist_decay_time[raw_decay]++;                  // Low 10 bits for decay time
                        datapoint++;
                }
                printf("ASCII -> ASCII\n");
                printf("Total Number of Photons, within the range (%g ns ~ %g ns) = %d\n",
                        g_DecayWinStart*1.0e9, g_DecayWinEnd*1.0e9, datapoint);

                fclose(fpTxt);
        }
        printf("Total Number of Photons, out of range = %d\n", out_of_range_count);
        printf("Total Number of Bad Photons (raw time gap <= %d) = %d\n", g_PhotonRemoval, bad_count);
        printf("Total Number of Error Photons (PMT 0) = %d\n", error_count);

        // Data Correction
        // 555 -> 11.1 ns (90Mhz), which means each step corresponds to 20 pico second.
        correct_with_offset(gHist_decay_time1, gHist_decay_time_corrected1, 0, 1024, g_TimingStep,
                gCorrections1, gPMT1_Offset);
        correct_with_offset(gHist_decay_time2, gHist_decay_time_corrected2, 0, 1024, g_TimingStep,
```

```
                gCorrections2, gPMT2_Offset);
        correct_with_offset(gHist_decay_time3, gHist_decay_time_corrected3, 0, 1024, g_TimingStep,
                gCorrections3, gPMT3_Offset);
        correct_with_offset(gHist_decay_time4, gHist_decay_time_corrected4, 0, 1024, g_TimingStep,
                gCorrections4, gPMT4_Offset);
        correct_with_offset(gHist_decay_time5, gHist_decay_time_corrected5, 0, 1024, g_TimingStep,
                gCorrections5, gPMT5_Offset);
        correct_with_offset(gHist_decay_time6, gHist_decay_time_corrected6, 0, 1024, g_TimingStep,
                gCorrections6, gPMT6_Offset);
        correct_with_offset(gHist_decay_time0, gHist_decay_time_corrected0, 0, 1024, g_TimingStep,
                gCorrections1, gPMT1_Offset);

        coeff = 1.0/g_LaserFrequency*1.0e9/g_TimingStep;
        for (i=0; i<g_TimingStep; i++) {
                graph_X[i] = coeff * (double)i;

                if ( (i >= DecayStart) && (i <= DecayEnd) ) {
                        gHist_decay_time_corrected[i] = gHist_decay_time_corrected1[i] +
                                gHist_decay_time_corrected2[i] + gHist_decay_time_corrected3[i] +
                                gHist_decay_time_corrected4[i] + gHist_decay_time_corrected5[i] +
                                gHist_decay_time_corrected6[i];
                                //+ gHist_decay_time_corrected0[i];
                        fprintf(fpHist, "%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n", graph_X[i],
                                gHist_decay_time_corrected[i], gHist_decay_time_corrected1[i],
                                gHist_decay_time_corrected2[i], gHist_decay_time_corrected3[i],
                                gHist_decay_time_corrected4[i], gHist_decay_time_corrected5[i],
                                gHist_decay_time_corrected6[i], gHist_decay_time_corrected0[i]);
                } else {
                        fprintf(fpHist, "%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n", graph_X[i], 0.0,
                                0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
                }
        }

        // Raw Data Histrogram
        if (g_isRawHistrogram == 1) {
                for (i=0; i<1024; i++)
                        fprintf(fpRawHist, "%u\t%u\t%u\t%u\t%u\t%u\t%u\t%u\t%u\n", i, gHist_decay_time[i],
                                gHist_decay_time1[i], gHist_decay_time2[i], gHist_decay_time3[i],
                                gHist_decay_time4[i], gHist_decay_time5[i], gHist_decay_time6[i],
                                gHist_decay_time0[i]);
        }

        fclose(fp);
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        fclose(fp4);
        fclose(fp5);
        fclose(fp6);
        fclose(fp0);
        fclose(fp_out_of_range);
        fclose(fp_bad);
        fclose(fpHist);
        fclose(fpRawHist);

        if (inBuffer)           free(inBuffer);
        if (graph_X)            free (graph_X);

        return 0;
}
```

```
//=============================================================================================
// Calculate RAW decay number
//=============================================================================================
int calc_raw_decay(unsigned int *raw_decay_start, unsigned int *raw_decay_end, double DecayStart, double
DecayEnd, partials **Corrections, int PMT_Offset, int TimingStep) {
	int decay_start, decay_end, i=-1;

	if ( (decay_start = (int)MAX(DecayStart, 0.0) + PMT_Offset) >= TimingStep)
		decay_start -= TimingStep;
	*raw_decay_start = Corrections[decay_start][0].index;

	if ( (decay_end = (int)MIN(DecayEnd, TimingStep-1) + PMT_Offset) >= TimingStep)
		decay_end -= TimingStep;
	while (Corrections[decay_end][++i].index != -1);
	*raw_decay_end = Corrections[decay_end][--i].index;

	if (*raw_decay_start < *raw_decay_end)
		return 1;
	else
		return 0;
}


//=============================================================================================
// Photon Count Analyze & Display
//=============================================================================================
int PhotonCountAnalyze_UNIX(void) {
	int *photon_count=NULL, *photon_count1=NULL, *photon_count2=NULL, *photon_count3=NULL,
		*photon_count4=NULL, *photon_count5=NULL, *photon_count6=NULL, photon_count0=NULL,
		BinSize, i, datapoint=0, err, BinNumber, PMTBuffer=0, decay;
	char buff[255];
	double *graph_X=NULL, reciprocal_LaserFrequency, reciprocal_resolution, start_time=0.0, end_time=0.0;
	double decay_corrected, Laser_tick;
	FILE *fpTxt;

	reciprocal_LaserFrequency = 1.0/g_LaserFrequency;
	reciprocal_resolution = 1.0/g_resolution;

	fpTxt = fopen(g_AnalyzedFile, "r");
	if (fpTxt == NULL)
		return -1;
	sprintf(buff, "[%s]\n", DateTimeToString());
	fseek(fpTxt, strlen(buff), SEEK_SET);

	// Determine photon start time and end time
	fscanf (fpTxt, "%lf\t%d\t%d\t%lf", &Laser_tick, &PMTBuffer, &decay, &decay_corrected);
	start_time = floor( (Laser_tick / g_LaserFrequency) / g_resolution ) * g_resolution;
	while (err=fscanf (fpTxt, "%lf\t%d\t%d\t%lf", &Laser_tick, &PMTBuffer, &decay, &decay_corrected) > 0);
	end_time = ceil( (Laser_tick / g_LaserFrequency) / g_resolution ) * g_resolution;
	printf("photon_start_time = %g sec\nphoton_end_time = %g sec\n", start_time, end_time);

	// initialization
	BinSize = (int)((end_time - start_time) / g_resolution);
	printf("BinSize = %d\n", BinSize);
	graph_X = (double*) malloc(BinSize * sizeof(double));
	photon_count = (int*) malloc(BinSize * sizeof(int));
	photon_count1 = (int*) malloc(BinSize * sizeof(int));
	photon_count2 = (int*) malloc(BinSize * sizeof(int));
	photon_count3 = (int*) malloc(BinSize * sizeof(int));
	photon_count4 = (int*) malloc(BinSize * sizeof(int));
	photon_count5 = (int*) malloc(BinSize * sizeof(int));
	photon_count6 = (int*) malloc(BinSize * sizeof(int));
```

```
photon_count0 = (int*) malloc(BinSize * sizeof(int));

for (i=0; i<BinSize; i++) {
	graph_X[i] = ((double)i + 0.5) * g_resolution + start_time;  graph_X[i] *= 1.0e3;// in milli-sec unit
	photon_count[i] = 0;
	photon_count1[i] = 0;
	photon_count2[i] = 0;
	photon_count3[i] = 0;
	photon_count4[i] = 0;
	photon_count5[i] = 0;
	photon_count6[i] = 0;
	photon_count0[i] = 0;
}

rewind(fpTxt);
sprintf(buff, "[%s]\n", DateTimeToString());
fseek(fpTxt, strlen(buff), SEEK_SET);

// Photon Count Histogram
while (err=fscanf (fpTxt, "%lf\t%d\t%d\t%lf", &Laser_tick, &PMTBuffer, &decay, &decay_corrected) > 0) {
	BinNumber = (int) ( (Laser_tick * reciprocal_LaserFrequency - start_time)*reciprocal_resolution );
	if ( BinNumber < 0 ) continue;
	if ( BinNumber >= BinSize ) break;

	photon_count[BinNumber]++;
	switch ( PMTBuffer ) {
		case 1 :   photon_count1[BinNumber]++;
			   break;
		case 2 :   photon_count2[BinNumber]++;
			   break;
		case 3 :   photon_count3[BinNumber]++;
			   break;
		case 4 :   photon_count4[BinNumber]++;
			   break;
		case 5 :   photon_count5[BinNumber]++;
			   break;
		case 6 :   photon_count6[BinNumber]++;
			   break;
		default :  photon_count0[BinNumber]++;
			   break;
	}
	datapoint++;
}
printf("Total Number of Photons Counted = %d\n", datapoint);
fclose(fpTxt);

// Histogram File Save
fpTxt = fopen(g_PhotonCountHisFile, "w");
for (i=0; i<BinSize; i++)
	fprintf(fpTxt, "%.15g\t%d\t\t%d\t%d\t%d\t%d\t%d\n", graph_X[i], photon_count[i],
		photon_count1[i], photon_count2[i], photon_count3[i], photon_count4[i],
		photon_count5[i], photon_count6[i]);
fclose(fpTxt);

if (graph_X) free (graph_X);
if (photon_count) free (photon_count);
if (photon_count1) free (photon_count1);
if (photon_count2) free (photon_count2);
if (photon_count3) free (photon_count3);
if (photon_count4) free (photon_count4);
if (photon_count5) free (photon_count5);
if (photon_count6) free (photon_count6);
```

```
	if (photon_count0) free (photon_count0);

	return 0;
}


//=============================================================================================
// Concatenate Photon counts
// Modified by SJK, at 11/12/2007 for clever peak finding
//=============================================================================================
int Concatenate_PhotonCountAnalysis_UNIX(void) {
	int err, PMTBuffer=0, decay;
	char buff[MAX_PATHNAME_LEN]={'\0'};
	char target_pathname[MAX_PATHNAME_LEN]={'\0'};
	double mstime1, mstime2, mstime_start, mstime_end;
	double decay_corrected, Laser_tick, Laser_tick_start, Laser_tick_end;
	double Laser_mstime, width, period;
	unsigned int count_total, count1, count2, count3, count4, count5, count6;
	FILE *fpTxt, *fpPhotonCount, *fpTarget, *fpPeak;
	double peak_time_weighted_count_sum, peak_search_begin_mstime, peak_search_end_mstime;
	unsigned int i, j, peak_count_sum, raw_peak_numbers, final_peak_numbers;
	double *mstime_array=NULL, *raw_peak_average_array=NULL, *peak_average_array=NULL;
	unsigned int *photon_count_array=NULL, *raw_peak_count_sum_array=NULL,
		*peak_count_sum_array=NULL, *peak_finding_array=NULL;
	unsigned int threshold;

	if (g_AnalyzedFile[0] == '\0') {
		printf("Error; Please analyze an ASCII unwrapped data first.\n");
		return -1;
	}

	// Reading (expected) width of the peak
	width = g_PC_PeakWidth;
	period = 1.0/(double)g_DropletFrequency;		// Droplet Generator Frequency (sec)

	// Redaing cursor positions & Bin numbers
	threshold = (unsigned int)g_PC_Threshold;

	mstime_start = MIN(g_PC_Start, g_PC_End) * 1.0e3;
	mstime_end = MAX(g_PC_Start, g_PC_End) * 1.0e3;

	// Open a Photon Count Histogram file
	fpPhotonCount = fopen(g_PhotonCountHisFile, "r");

	// Open a Peak List file
	strncpy(buff, g_ConcatenatedFile, strlen(g_ConcatenatedFile)-4);
	buff[strlen(g_ConcatenatedFile)-4] = '\0';
	strcat(buff, "_peaks.txt");
	fpPeak = fopen(buff, "w");

	// Initialization
	raw_peak_numbers = 0;
	peak_search_begin_mstime = mstime_start - 0.5*width*1.0e3;
	peak_search_end_mstime = mstime_end + 0.5*width*1.0e3;

	// Load all peaks (over the threashold) into the memory
	while ( err = fscanf (fpPhotonCount, "%lf\t%d\t\t%d\t%d\t%d\t%d\t%d\t%d", &Laser_mstime, &count_total,
	&count1, &count2, &count3, &count4, &count5, &count6) > 0 ) {
		if ((Laser_mstime >= peak_search_begin_mstime) &&
		(Laser_mstime<=peak_search_end_mstime) ) {
			if (count_total > threshold) {		// Over the Threshold
				mstime_array = (double*) realloc( mstime_array,
```

```
                                        (++raw_peak_numbers)*sizeof(double) );
                                photon_count_array = (unsigned int*) realloc( photon_count_array,
                                        raw_peak_numbers*sizeof(unsigned int) );
                                mstime_array[raw_peak_numbers-1] = Laser_mstime;
                                photon_count_array[raw_peak_numbers-1] = count_total;
                        }
                }
                if ( Laser_mstime > peak_search_end_mstime )    // Exit the loop
                        break;
}

// if not enough initial peak has been detected
if (raw_peak_numbers < 9) {
                printf("Error; Please lower the threshold cursor to detect a reasonable number of photon count
                        peaks\n");
                goto error;
}

// Initialization
peak_count_sum = 0;
peak_time_weighted_count_sum = 0.0;

peak_search_begin_mstime = mstime_start - width*1.0e3;
peak_search_end_mstime = mstime_start + period*1.0e3;
final_peak_numbers = 0;

raw_peak_average_array = (double*) malloc(raw_peak_numbers * sizeof(double));
raw_peak_count_sum_array = (unsigned int*) malloc(raw_peak_numbers * sizeof(int));

// Calculate average peaks assigned for every raw peak over the threashold
for (i=0; i<raw_peak_numbers; i++) {

                mstime1 = mstime_array[i];
                mstime2 = mstime1 + width*1.0e3;

                // Calculate peak sum (density) within the range (width)
                for (j=i; j<raw_peak_numbers; j++) {
                        if ( mstime_array[j] <= mstime2) {
                                peak_time_weighted_count_sum += mstime_array[j] *
                                        (double)photon_count_array[j];
                                peak_count_sum += photon_count_array[j];
                        } else {
                                break;
                        }
                }

                // if there were peaks in this range
                if (peak_count_sum > 0)
                        raw_peak_average_array[i] = peak_time_weighted_count_sum/(double)peak_count_sum;
                else
                        raw_peak_average_array[i] = 0.0;

                raw_peak_count_sum_array[i] = peak_count_sum;
                peak_time_weighted_count_sum = 0.0;
                peak_count_sum = 0;
}

// Finding Peaks
peak_finding_array = (unsigned int*) malloc(raw_peak_numbers * sizeof(int));
peak_finding_array[2] = raw_peak_count_sum_array[0] + raw_peak_count_sum_array[1] +
                raw_peak_count_sum_array[2] + raw_peak_count_sum_array[3] + raw_peak_count_sum_array[4];
```

```
for (i=3; i<raw_peak_numbers-2; i++)
        peak_finding_array[i] = peak_finding_array[i-1] + raw_peak_count_sum_array[i+2] -
                raw_peak_count_sum_array[i-3];

for (i=4; i<raw_peak_numbers-4; i++) {
        if ( (peak_finding_array[i] >= peak_finding_array[i-2]) &&
         (peak_finding_array[i] >= peak_finding_array[i-1]) &&
         (peak_finding_array[i] > peak_finding_array[i+1]) &&
         (peak_finding_array[i] > peak_finding_array[i+2]) ) {

                peak_average_array = (double*) realloc( peak_average_array,
                        (++final_peak_numbers)*sizeof(double) );
                peak_count_sum_array = (unsigned int*) realloc( peak_count_sum_array,
                        final_peak_numbers*sizeof(int) );

                peak_average_array[final_peak_numbers-1] = raw_peak_average_array[i];
                peak_count_sum_array[final_peak_numbers-1] = raw_peak_count_sum_array[i];
        }
}

// If you found peaks!
if ( final_peak_numbers > 0 ) {
        // Print peaks
        for (i=0; i<final_peak_numbers; i++) {
                fprintf(fpPeak, "%.15lg\t%u\n", peak_average_array[i], peak_count_sum_array[i]);
        }

        // Calcuation of initial laser tick range
        Laser_tick_start = (double)( (unsigned int)( (peak_average_array[0]-0.5*width*1.0e3)*1.0e-3 /
                g_resolution - 0.5 ) ) * g_resolution * g_LaserFrequency;
        Laser_tick_end = (double)( (unsigned int)( (peak_average_array[0]+0.5*width*1.0e3)*1.0e-3 /
                g_resolution + 0.5 ) ) * g_resolution * g_LaserFrequency + 1.0;

        // Open an Unwrapped Laser Tick file
        fpTxt = fopen(g_AnalyzedFile, "r");
        sprintf(buff, "[%s]\n", DateTimeToString());
        fseek(fpTxt, strlen(buff), SEEK_SET);

        // Open a concaternated Target File & Write Date and Time
        fpTarget = fopen(g_ConcatenatedFile, "w");
        fprintf(fpTarget, "[%s]\n", DateTimeToString());

        // Concaternation
        i=0;
        while ( err = fscanf (fpTxt, "%lf\t%d\t%d\t%lf", &Laser_tick, &PMTBuffer, &decay,
        &decay_corrected) > 0 ) {
                if ( Laser_tick >= Laser_tick_start )
                        fprintf(fpTarget, "%.15lg\t%u\t%04u\t%04lf\n", Laser_tick, PMTBuffer, decay,
                                decay_corrected);

                if ( Laser_tick > Laser_tick_end ) {
                        if ( ++i > (final_peak_numbers-1) )
                                break;

                        Laser_tick_start = (double)( (unsigned int)( (peak_average_array[i]-
                                0.5*width*1.0e3)*1.0e-3 / g_resolution - 0.5 ) ) * g_resolution *
                                g_LaserFrequency;
                        Laser_tick_end = (double)( (unsigned int) ( (peak_average_array[i] +
                                0.5*width*1.0e3)*1.0e-3 / g_resolution + 0.5 ) ) * g_resolution *
                                g_LaserFrequency + 1.0;
                }
        }
```

```
                fclose(fpTarget);
                fclose(fpTxt);
        } else {
                printf("Error; No Peaks found!");
        }

error:
        fclose(fpPeak);
        fclose(fpPhotonCount);

        if (mstime_array)
                free (mstime_array);
        if (photon_count_array)
                free (photon_count_array);
        if (peak_average_array)
                free (peak_average_array);
        if (raw_peak_average_array)
                free (raw_peak_average_array);
        if (raw_peak_count_sum_array)
                free (raw_peak_count_sum_array);
        if (peak_count_sum_array)
                free (peak_count_sum_array);
        if (peak_finding_array)
                free (peak_finding_array);

        return 0;
}


//=============================================================================================
// Selective Photon Count Analyze
//=============================================================================================
int Selective_PhotonCountAnalyze_UNIX(void) {
   int BinNumber1, BinNumber2;
   FILE *fpTxt, *fpTarget;
        char buff[255], buff2[255], *pFilename;
        double decay_corrected, Laser_tick, Laser_tick1, Laser_tick2;
        int err, PMTBuffer=0, decay;

        if (g_AnalyzedFile[0] == '\0') {
                printf("Error; Please analyze an ASCII unwrapped data first.\n");
                return -1;
        }

        BinNumber1 = (int)( MIN(g_PC_Start, g_PC_End) / g_resolution - 0.5 );
        BinNumber2 = (int)( MAX(g_PC_Start, g_PC_End) / g_resolution + 0.5 );

        Laser_tick1 = (double)BinNumber1 * g_resolution * g_LaserFrequency;
        Laser_tick2 = (double)BinNumber2 * g_resolution * g_LaserFrequency + 1.0;

        if ( (fpTxt = fopen(g_AnalyzedFile, "r")) == NULL )
                return -1;
        sprintf(buff, "[%s]\n", DateTimeToString());
        fseek(fpTxt, strlen(buff), SEEK_SET);

        // By Default, in case that there is no directory information included.
        sprintf(buff, "selective_%s", g_AnalyzedFile);

        pFilename = strrchr( g_AnalyzedFile, '\\');                     // for Windows
        if (pFilename != NULL) {
                pFilename++;
```

```
		strncpy(buff2, g_AnalyzedFile, (int)(pFilename - g_AnalyzedFile));
		sprintf(buff, "%sselective_%s", buff2, pFilename);
	}

	pFilename = strrchr( g_AnalyzedFile, '/');                    // for UNIX
	if (pFilename != NULL) {
		pFilename++;

		strncpy(buff2, g_AnalyzedFile, (int)(pFilename - g_AnalyzedFile));
		sprintf(buff, "%sselective_%s", buff2, pFilename);
	}

	if ( (fpTarget = fopen(buff, "w")) == NULL )
		return -2;
	fprintf(fpTarget, "[%s]\n", DateTimeToString());

	while (err=fscanf (fpTxt, "%lf\t%d\t%d\t%lf", &Laser_tick, &PMTBuffer, &decay, &decay_corrected) > 0) {
		if ( Laser_tick >= Laser_tick1 )
			fprintf(fpTarget, "%.15lg\t%u\t%04u\t%04lf\n", Laser_tick, PMTBuffer, decay,
				decay_corrected);

		if ( Laser_tick > Laser_tick2 )
			break;
	}
	fclose(fpTarget);
	fclose(fpTxt);

	return 0;
}
```

References

1. Nelson, D.L. and M.M. Cox, *Lehninger Principles of Biochemistry*. 3rd ed. 2000. Chapter 5-7.
2. Blake, C. and L. Serpell, *Synchrotron X-ray studies suggest that the core of the transthyretin amyloid fibril is a continuous beta-sheet helix.* Structure, 1996. **4**: p. 989-998
3. Brown, T.L., *Making Truth: Metaphor in Science*. 2003: University of Illinois Press. Chapter 7-Protein Folding
4. Huang, G.S. and T.G. Oas, *Submillisecond Folding of Monomeric λ Repressor.* Proc. Natl. Acad. Sci. USA, 1995. **92**: p. 6878-6882
5. Anfinsen, C.B., E. Haber, M. Sela, and J. F. H. White, *The Kinetics of Formation of Native Ribonuclease During Oxidation of the Reduced Polypeptide Chain.* Proc. Natl. Acad. Sci. USA, 1961. **47**(9): p. 1309-1314.
6. Levinthal, C., *Are there pathways for protein folding?* J. Chim. Phys, 1968. **65**: p. 44-45
7. Levinthal, C. and A. Rawitch. *How to fold graciously*. in *Mossbauer Spectroscopy in Biological Systems*. 1969. Monticello, Illinois: University of Illinois at Urbana-Champaign.
8. Gruebele, M., *The Fast Protein Folding Problem.* Annu. Rev. Phys. Chem, 1999. **50**: p. 485-516
9. Snow, C., H. Nguyen, V. Pande, and M. Gruebele, *Absolute comparison of simulated and experimental protein folding dynamics.* Nature, 2002. **420**: p. 102-106.
10. Bryngelson, J.D., J.N. Onuchic, N.D. Socci, and P.G. Wolynes, *Funnels, Pathways, and the Energy Landscape of Protein Folding: A Synthesis.* Proteins: Struct., Funct., Genet, 1995. **21**: p. 167-195
11. Onuchic, J.N., Z. Luthey-Schulten, and P.G. Wolynes, *Theory of Protein Folding: The Energy Landscape Perspective.* Annu. Rev. Phys. Chem., 1997. **48**: p. 545-600.

12. Guinier, A. and G. Fournet, *Small-Angle Scattering of X-rays*. 1955, New York: John Wiley & Sons, Inc.
13. Dumont, C., Y. Matsumura, S.J. Kim, J. Li, E. Kondrashkina, H. Kihara, and M. Gruebele, *Solvent-tuning collapse and helix formation time scales of lambda6-85.* Protein Science, 2006. **15**: p. 2596-2604.
14. Johnson, W.C.J., *Circular Dichroism Instrumentation*, in *Circular Dichroism and the Conformational Analysis of Biomolecules*, G.D. Fasman, Editor. 1996, Plenum Press: New York. p. 635-652.
15. Rhoades, E., E. Gussakovsky, and G. Haran, *Watching proteins fold one molecule at a time.* Proc Natl Acad Sci U S A, 2003. **100**(6): p. 3197-202.
16. Rhoades, E., M. Cohen, B. Schuler, and G. Haran, *Two-state folding observed in individual protein molecules.* J. AM. Chem. Soc., 2004. **126**: p. 14686-14687.
17. Tardieu, A., A. Le Verge, M. Malfois, F. Bonneté, S. Finet, M. Riés-Kautt, and L. Belloni, *Proteins in solution : from X-ray scattering intensities to interaction potentials.* J. Cryst. Growth, 1999. **196**: p. 193-203.
18. Javid, N., K. Vogtt, C. Krywka, M. Tolan, and R. Winter, *Protein-protein interactions in complex cosolvent solutions.* Phys. Chem. Chem. Phys, 2007. **8**: p. 679-689.
19. Ebbinghaus, S., S.J. Kim, M. Heyden, X. Yu, U. Heugen, M. Gruebele, D. Leitner, and M. Havenith, *An extended dynamical hydration shell around proteins.* Proc. Nat. Acad. Sci. USA, 2007. **104**: p. 20749-20752.
20. Yang, W.Y. and M. Gruebele, *Folding at the speed limit.* Nature, 2003. **423**(6936): p. 193-197.
21. Silow, M., Y. Tan, A.R. Fersht, and M. Oliveberg, *Formation of Short-Lived Protein Aggregates Directly from the Coil in Two-State Folding.* Biochemistry, 1999. **38**: p. 13006-13012.
22. Yang, W.Y. and M. Gruebele, *Binary and ternary aggregation with tethered protein constructs.* Biophys. J., 2006. **90**: p. 2930-2937.
23. Otzen, D.E., S. Miron, M. Akke, and M. Oliveberg, *Transient aggregation and stable dimerization induced by introducing an Alzheimer sequence into a water-soluble protein* Biochemistry, 2004. **43**(41): p. 12964-12978.

24. Niehbur, M. and M.H.J. Koch, *Effects of Urea and Trimethylamine-N-Oxide (TMAO) on the Interactions of Lysozyme in Solution.* Biophys. J., 2005. **89**: p. 1978-1983.

25. Ghaemmaghami, S., J.M. Word, R.E. Burton, J.S. Richardson, and T.G. Oas, *Folding kinetics of a fluorescent variant of monomeric lambda repressor.* Biochemistry, 1998. **37**(25): p. 9179-9185.

26. Larson, S.M., A.A.D. Nardo, and A.R. Davidson, *Analysis of covariation in an SH3 domain sequence alignment: Applications in tertiary contact prediction and the design of compensating hydrophobic core substitutions.* J. Mol. Biol., 2000. **303**(3): p. 433-446.

27. Edelhoch, H., *Spectroscopic Determination of Tryptophan and Tyrosine in Proteins.* Biochemistry, 1967. **6**(7): p. 1948-1954.

28. Fischetti, R., S. Stepanov, G. Rosenbaum, R. Barrea, E. Black, D. Gore, R. Heurich, E. Kondrashkina, A.J. Kropf, S. Wang, K. Zhang, T.C. Irving, and G.B. Bunke, *The BioCAT undulator beamline 18ID: a facility for biological non-crystalline diffraction and X-ray absorption spectroscopy at the Advanced Photon Source.* J. Synchrotron Rad., 2004. **11**: p. 399-405.

29. Larios, E., *a Computational-Experimental Study of Small Globular Proteins*, in *Physics Ph.D. Thesis*. 2005, University of Illinois at Urbana-Champaign.

30. Metropolis, N., A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, *Equation of State Calculations by Fast Computing Machines.* J. Chem. Phys., 1953. **21**: p. 1087-1092.

31. Chandler, D., *Modern Statistical Mechanics*. 1989, Oxford: Oxford University Press.

32. Rayleigh, L., *On the diffraction of light by spheres of small relative index.* Proc. Roy. Soc. (London), 1914. **A-90**: p. 219-225.

33. Rayleigh, L., *The incidence of light upon a transparent sphere of dimension comparable to the wavelength.* Proc. Roy. Soc. (London), 1910. **A-84**: p. 25-46.

34. Suhonen, H., *Simulation of Small-Angle X-Ray Scattering from Collagen Fibrils*, in *Master's Thesis*. 2005, University of Helsinki.

35. Moré, J.J., B.S. Garbow, and K.E. Hillstrom, *User Guide for MINPACK-1*, in

Argonne National Laboratory Report. 1980, Argonne National Laboratory: Argonne. p. 1-48.

36. Moré, J.J., D.C. Sorensen, and K.E. Hillstrom, *The MINPACK Project*, in *Sources and Development of Mathematical Software*, W.J. Cowell, Editor. 1984, Prentice-Hall: New York. p. 88-111.
37. Humphrey, W.F., A. Dalke, and K. Schulten, *VMD Visual Molecular Dynamics.* Journal of Molecular Graphics, 1996. **14**: p. 33-38.
38. Svergun, D., C. Barberato, and M.H.J. Koch, *CRYSOL: a Program to Evaluate X-ray Solution Scattering of Biological Macromolecules from Atomic Coordinates.* J. Appl. Cryst., 1995. **28**: p. 768-773.
39. Svergun, D., S. Richard, M.H.J. Koch, Z. Sayers, S. Kuprin, and G. Zaccai, *Hydration shell in solution: validation by x-ray and neutron scattering.* Proc. Nat. Acad. Sci. USA, 1998. **95**: p. 2267-2272.
40. Narayanan, J. and X.Y. Liu, *Protein Interactions in Undersaturated and Supersaturated Solutions: A Study Using Light and X-Ray Scattering.* Biophys. J., 2003. **84**: p. 523-532.
41. Bosdevant, N., D. Borgis, and T. Ha-Duong, *A coarse-grained protein-protein potential derived from an all-atom force field.* J. Phys. Chem. B, 2007. **111**(31): p. 9390-9399.
42. Vidugiris, G.J.A., J.L. Markley, and C.A. Royer, *Evidence for a Molten Globule-like Transition State in Protein Folding from Determination of Activation Volumes.* Biochemistry, 1995. **34**: p. 4909-4912.
43. Khorasanizadeh, S., I. Peters, and H. Roder, *Evidence for a Three-State Model of Protein Folding from Kinetic Analysis of Ubiquitin Variants with Altered Core Residues.* Nat. Struct. Biol., 1996. **3**(2): p. 193-205.
44. Wolde, P.R.t. and D. Chandler, *Drying-induced hydrophobic polymer collapse.* J. Mol. Biol., 2002. **311**: p. 373-393.
45. Fujiyoshi, Y., K. Mitsuoka, B.L. de Groot, A. Philippsen, H. Grubmuller, P. Agre, and A. Engel, *Structure and Function of Water Channels.* Curr. Op. Struct. Biol., 2002. **12**(4): p. 509-515.
46. Ernst, J.A., R.T. Clubb, H.-X. Zhou, A.M. Gronenborn, and G.M. Clore,

Demonstration of Positionally Disordered Water Within a Protein Hydrophobic Cavity by NMR. Science, 1995. **267**: p. 1813-1817.

47. Fenimore, P.W., H. Frauenfelder, B.H. McMahon, and F.G. Parak, *Slaving: Solvent fluctuations dominate protein dynamics and functions.* Proc. Nat. Acad. Sci. USA, 2002. **99**(25): p. 16047-16051.
48. Ansari, A., J. Berendzen, S.F. Bowne, H. Frauenfelder, I.E.T. Iben, T.B. Sauke, E. Shyamsunder, and R.D. Young, *Protein States and Protein Quakes.* Proc. Nat. Acad. Sci. USA, 1985. **82**(15): p. 5000-5004.
49. Despa, F., A. Fernández, and R.S. Berry, *Dielectric modulation of biological water.* Phys. Rev. Lett., 2004. **93**: p. 228104.
50. Burling, F.T., W.I. Weis, K.M. Flaherty, and A.T. Brunger, *Direct observation of protein solvation and discrete disorder with experimental crystallographic phases.* Science, 1996. **271**(5245): p. 72-77.
51. Collins, M., G. Hummer, M.L. Quillin, B.W. Matthews, and S.M. Gruner, *Cooperative water filling of a nonpolar portein cavity observed by high pressure crystallography and simulation.* Proc. Nat. Acad. Sci. USA, 2005. **102**(46): p. 16668-16671.
52. Grant, E.H., R.J. Sheppard, and G.P. South, *Dielectric behaviour of biological molecules in solution (Clarendon, Oxford).* 1978, Oxford: Clarendon.
53. Kuwata, K., R. Shastry, H. Cheng, M. Hoshino, C.A. Batt, Y. Goto, and H. Roder, *Structural and kinetic characterization of early folding events in beta-lactoglobulin.* Nat. Struc. Bio., 2001. **8**(2): p. 151-155.
54. Siegrist, K., C.R. Bucher, I. Mandelbaum, A.R. Walker, R. Balu, S.K. Gregurick, and D.F. Plusquellic, *High-Resolution Terahertz Spectroscopy of Crystalline Trialanine: Extreme Sensitivity to Beta-Sheet Structure and Cocrystallized Water.* J. AM. Chem. Soc., 2006. **128**: p. 5764-5775.
55. Zanotti, J.M., M.C. Bellissent-Funel, and J. Parello, *Hydration-coupled dynamics in proteins studied by neutron scattering and NMR: The case of the typical EF-hand calcium-binding parvalbumin.* Biophys. J., 1999. **76**(5): p. 2390-2411.
56. Hochstrasser, R.M., *Techniques - Spectroscopy at a stretch.* Nature, 2005. **434**(7033): p. 570-571.

57. Zhong, D.P., S.K. Pal, D.Q. Zhang, S.I. Chan, and A.H. Zewail, *Femtosecond dynamics of rubredoxin: Tryptophan solvation and resonance energy transfer in the protein.* Proc. Nat. Acad. Sci. USA, 2002. **99**(1): p. 13-18.

58. Ronne, C., L. Thrane, P.-O. Astrand, A. Wallqvist, K.V. Mikkelsen, and S.R. Keiding, J. Chem. Phys., 1997. **107**: p. 5319-5330.

59. Beard, M.C., G.M. Turner, and C.A. Schmuttenmaer, *Terahertz spectroscopy.* Journal of Physical Chemistry B, 2002. **106**(29): p. 7146-7159.

60. Bergner, A., U. Heugen, E. Brundermann, G. Schwaab, M. Havenith, D.R. Chamberlin, and E.E. Haller, *New p-Ge THz laser spectrometer for the study of solutions: THz absorption spectroscopy of water.* Rev. Sci. Inst., 2005. **76**(6): p. 063110.

61. Chen, J.-Y., J.R. Knab, J. Cerne, and A.G. Markelz, *Large oxidation dependence observed in terahertz dielectric response for cytochrome c.* Phys. Rev. E, 2005. **72**: p. 040901(1-4).

62. Xu, J., K.W. Plaxco, and S.J. Allen, *Probing the collective vibrational dynamics of a protein in liquid water by terahertz absorption spectroscopy.* Prot. Sci., 2006. **15**: p. 1175-1181.

63. Leitner, D.M., M. Havenith, and M. Gruebele, *Biomolecule large-amplitude motion and solvation dynamics: modeling and probes from THz to X-rays.* Int. Rev. Phys. Chem., 2006. **25**(4): p. 553-582.

64. Ebbinghaus, S., *THz Spectroscopy of Biomolecules*, in *Ph.D. Thesis in Chemistry and Biochemistry*. 2007, Ruhr-University-Bochum: Bochum, Germany.

65. Whitmire, S.E., D. Wolpert, A.G. Markelz, J.R. Hillebrecht, J. Galan, and R.R. Birge, *Protein Flexibility and Conformational State: A Comparison of Collective Vibrational Modes of Wild-Type and D96N Bacteriorhodopsin.* Biophys. J., 2003. **85**: p. 1269-1277.

66. Ervin, J., E. Larios, S. Osvath, K. Schulten, and M. Gruebele, *What causes hyperfluorescence: Folding intermediates or conformationally flexible native states?* Biophys. J., 2002. **83**(1): p. 473-483.

67. Glatter, O. and O. Kratky, eds. *Small Angle X-ray Scattering*. 1982, Academic Press.

68. DiStefano, D.L. and A.J. Wand, *Two-Dimensional ^{1}H NMR Study of Human Ubiquitin: A Main Chain Directed Assignment and Structure Analysis.* Biochemistry, 1987. **26**: p. 7272-7281.

69. Laub, P., S. Khorisanizadeh, and H. Roder, *Localized Solution Structure Refinement of an F45W Variant of Ubiquitin Using Stochastic Boundary Molecular Dynamics and NMR Distance Restraints.* Protein Sci., 1995. **4**: p. 973-982.

70. Fan, H., A.E. Mark, J. Zhu, and B. Honig, Proc. Nat. Acad. Sci. USA, 2005. **102**(19): p. 6760.

71. Qin, Z., J. Ervin, E. Larios, M. Gruebele, and H. Kihara, *Formation of a compact structured ensemble without fluorescence signature early during ubiquitin folding.* J. Phys. Chem. B, 2002. **106**(50): p. 13040-13046.

72. Lazar, G.A., J.R. Desjarlais, and T.M. Handel, *De Novo Design Of the Hydrophobic Core Of Ubiquitin.* Protein Sci., 1997. **6**(6): p. 1167-1178.

73. Fischer, H., I. Polikarpov, and A.F. Craievich, *Average protein density is a molecular-weight-dependent function.* Prot. Sci., 2004. **13**: p. 2825-2828.

74. Heugen, U., G. Schwaab, E. Brundermann, M. Heyden, X. Yu, D.M. Leitner, and M. Havenith, *Solute induced retardation of water dynamics: Hydration water probed directly by THz spectroscopy.* Proc. Natl. Acad. Sci. USA, 2006. **103**: p. 12301-12306.

75. Choudhury, N. and B.M. Pettitt, *Dynamics of Water Trapped between Hydrophobic Solutes.* J. Phys. Chem. B, 2005. **109**: p. 6422.

76. Makarov, V.A., M. Feig, B.K. Andrews, and B.M. Pettitt, *Diffusion of Solvent Around Biomolecular Solutes. A Molecular Dynamics Simulation Study.* Biophys. J., 1998. **75**: p. 150-158.

77. Ebbinghaus, S., S.J. Kim, M. Heyden, X. Yu, M. Gruebele, D. Leitner, and M. Havenith, *Protein sequence- and pH-dependent hydration probed by Terahertz spectroscopy.* J. AM. Chem. Soc., 2008. **130**(8): p. 2374-2375.

78. Chakraborty, S., S.K. Sinha, and S. Bandyopadhyay, J. Phys. Chem. B, 2007. **111**: p. 13626-13631.

79. Pal, S.K., J. Peon, B. Bagchi, and A.H. Zewail, *Biological Water: Femtosecond*

Dynamics of Macromolecular Hydration. J . Phys. Chem. B, 2002. **106**: p. 12376.

80. Russo, D., R.K. Muraka, G. Hura, E. Verschell, T. Copley, and J.R.D. Head-Gordon, *Evidence for anomalous hydration dynamics as a function of temperature near a model hydrophobic peptide.* J. Phys. Chem. B, 2004. **108**: p. 19885.
81. Bandyopadhyay, S., S. Chakraborty, and B. Bagchi, *Coupling between hydration layer dynamics and unfolding kinetics of HP-36.* J. Chem. Phys, 2006. **125**: p. 84912.
82. Yang, W.Y. and M. Gruebele, *Rate-temperature relationships in lambda repressor fragment 6-85 folding.* Biochemistry, 2004. **43**: p. 13018-13025.
83. Born, B., S.J. Kim, M. Gruebele, and M. Havenith, *The THz dance of water with the proteins: The effect of protein flexibility on the dynamical hydration shell of ubiquitin.* Faraday Discussion 141: Water – From Interfaces to the Bulk, 2008. **in press**.
84. Golosov, A.A. and M.J. Karplus, J. Phys. Chem. B, 2007. **111**: p. 1482-1490.
85. Berman, H.M., J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne, *The Protein Data Bank.* Nucleic Acids Research, 2000. **28**(1): p. 235-242.
86. Jacob, M.H., C. Saudan, G. Holtermann, A. Martin, D. Perl, A.E. Merbach, and F.X. Schmid, *Water contributes actively to the rapid crossing of a protein unfolding barrier.* Journal of Molecular Biology, 2002. **318**(3): p. 837-845.
87. Schroder, C., T. Rudas, S. Boresch, and O. Steinhauser, *Simulation studies of the protein-water interface. I. Properties at the molecular resolution.* Journal of Chemical Physics, 2006. **124**(23): p. 234907.
88. Knab, J., J.Y. Chen, and A. Markelz, Biophys. J., 2006. **90**: p. 2576.
89. Dexheimer, S., *Terahertz spectroscopy: principles and applications*. 2007, London: Taylor & Francis.
90. Rosenfeld, D.E. and C.A. Schmuttenmaer, *Dynamics of water confined within reverse micelles.* J. Phys. Chem. B., 2007. **110**: p. 14304-14312.
91. Plusquellic, D.F., K. Siegrist, E.J. Heilweil, and O. Esenturk, *Applications of Terahertz Spectroscopy in Biosystems.* Chem Phys Chem, 2007. **8**: p. 2412-2431.
92. Schmuttenmaer, C.A., *Exploring dynamics in the far infrared with terahertz*

spectroscopy. Chem. Rev., 2004. **104**: p. 1759-1780.

93. Groma, G.I., J. Hebling, I.Z. Kozma, G. Varo, J. Hauer, J. Kuhl, and E. Riedle, *Terahertz radiation from bacteriorhodopsin reveals correlated primary electron and proton transfer processes.* Proc. Nat. Acad. Sci. USA, 2008. **105**(19): p. 6888-6893.
94. Liu, H.-B. and X.-C. Zhang, *Dehydration kinetics of D-glucose monohydrate studied using THz time-domain spectroscopy.* Chem. Phys. Lett., 2006. **429**: p. 229-233.
95. Kim, S.J., B. Born, M. Havenith, and M. Gruebele, *Real-time detection of protein-water dynamics upon protein folding by Terahertz absorption spectroscopy.* Angewandte Chemie International Edition, 2008. **47**(34): p. 6486-6489 (as an inside cover story).
96. Khorasanizadeh, S., I. Peters, T. Butt, and H. Roder, *Folding and Stability of a Tryptophan-Containing Mutant of Ubiquitin.* Biochemistry, 1993. **32**: p. 7054-7063.
97. Larios, E., J.S. Li, K. Schulten, H. Kihara, and M. Gruebele, *Multiple probes reveal a native-like intermediate during low-temperature refolding of ubiquitin.* J. Mol. Biol., 2004. **340**: p. 115-125.
98. Tani, M., M. Herrmann, and K. Sakai, *Generation and detection of terahertz pulsed radiation with photoconductive antennas and its application to imaging.* Meas. Sci. Technol, 2002. **13**: p. 1739-1745
99. Kramers, H.A., *Brownian Motion In A Field of Force and the Diffusion Model of Chemical Reactions.* Physica, 1940. **7**(4): p. 284.
100. Hagen, S.J., J. Hofrichter, A. Szabo, and W.A. Eaton, *Diffusion-limited Contact Formation in Unfolded Cytochrome c: Estimating the Maximum Rate of Protein Folding.* Proc. Natl. Acad. Sci. USA, 1996. **93**: p. 11615-11617.
101. Leitner, D., M. Gruebele, and M. Havenith, *Solvation dynamics of biomolecules: modeling and terahertz experiments.* perspectives HFSP Journal, 2008. **in press**.
102. Sadqi, M., D. Fushman, and V. Muñoz, *Atom-by-atom analysis of global downhill protein folding.* Nature, 2006. **442**: p. 317.
103. Liphardt, J., B. Onoa, S.B. Smith, I. Tinoco, and C. Bustamante, *Reversible*

unfolding of single RNA molecules by mechanical force. Science, 2001. **292**(5517): p. 733-737.

104. Schuler, B., *Single-Molecule Fluorescence Spectroscopy of Protein Folding.* ChemPhysChem, 2005. **6**: p. 1206-1220.
105. Schuler, B., E.A. Lipman, and W.A. Eaton, *Probing the free-energy surface for protein folding with single-molecule fluorescence spectroscopy.* Nature, 2002. **419**: p. 743-747.
106. Fernandez, J.M. and H. Li, *Force-clamp spectroscopy monitors the folding trajectory of a single protein.* Science, 2004. **303**(5664): p. 1674-1678.
107. Ashkin, A., *Acceleration and Trapping of Particles by Radiation Pressure.* Physical Review Letters, 1970. **24**(4): p. 156-159.
108. Ashkin, A. and J.M. Dziedzic, *Optical levitation by radiation pressure.* Appl. Phys. Lett., 1971. **19**: p. 283-285.
109. Roosen, G. and C. Imbert, *Optical levitation by means of two horizontal laser beams: a theoretical and experimental study.* Phys. Lett., 1976. **59A**: p. 6-8.
110. Ashkin, A., J.M. Dziedzic, J.E. Bjorkholm, and S. Chu, *Observation of a single-beam gradient force optical trap for dielectric particles.* Optics Letters, 1986. **11**: p. 5.
111. Svoboda, K. and S.M. Block, *Biological applications of optical forces.* Annu. Rev. Biophys. and Biomolec. Structure, 1994. **23**: p. 247-285.
112. Neuman, K.C. and S.M. Block, *Optical Trapping.* Rev. Sci. Instrum., 2004. **75**(9): p. 2788.
113. Lee, E.R., *Microdrop Generation*. 2003: CRC Press.
114. Ulmke, H., T. Wriedt, and K. Bauckhage, *Piezoelectric Droplet Generator for the Calibration of Particle-Sizing Instruments.* Chem. Eng. Technol, 2001. **24**: p. 3.
115. MicroFab Technologies, *JetDrive™ III User's Guide*. 2003, MicroFab Technologies, Inc.: Plano, TX. p. 1-5.
116. MicroFab Technologies, *JetDrive™ III Command Set, Version 5.3*. 2003, MicroFab Technologies, Inc.: Plano, Texas. p. 1-12.
117. Odde, D.J. and M.J. Renn, *Laser-guided direct writing for applications in biotechnology.* Trends in Biotechnology, 1999. **17**: p. 385–389.

118. Nahmias, Y.K. and D.J. Odde, *Analysis of Radiation Forces in Laser Trapping and Laser-Guided Direct Writing Applications.* IEEE Journal of Quantum Electronics, 2002. **38**(2).
119. *NI 622x Specifications*, National Instruments.
120. *DAQ: 653X User Manual*. 2001, National Instruments: austin, Texas.
121. http://www.ni.com. [cited.
122. Microsoft. *MSDN Library 2005*. [cited.
123. van de Hulst, *Light Scattering by Small Particles*. 1981, New York: Dover Publications, Inc.
124. Mie, G., *Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen.* Ann. Physik, 1908. **25**: p. 377.
125. Debye, P., *Der Lichtdruck auf Kugeln von Beliebigem Material.* Ann. Phys., 1909. **30**: p. 57-136.
126. Lorenz, L., *Lysbevaegelsen i og uden for en haf plane lysbolger belyst kulge.* Videnskab. selskab. skrifter, 1890. **6**: p. 1-62.
127. Hecht, E., *Optics*. 2001: Addison Wesley.
128. Gouesbet, G. and G. Grehan, *Sur la generalisation de la theoriede Lorenz-Mie.* J. Opt. (Paris), 1982. **13**: p. 97-103.
129. Gouesbet, G., B. Maheu, and G. Grehan, *Light scattering from a sphere arbitrarily located in a Gaussian beam, using a Bromwich formulation.* J. Opt. Soc. Am. A, 1988. **5**(9): p. 1427.
130. Maheu, B., G. Gouesbet, and G. Grehan, *A concise presentation of the generalized Lorenz-Mie theory for arbitrary location of the scatterer in an arbitrary incident profile.* J. Opt. (Paris), 1988. **19**: p. 59-67.
131. Gouesbet, G., *Validity of the localized approximation for arbitrary shaped beams in the generalized Lorenz–Mie theory for spheres.* J. Opt. Soc. Amer. A,, 1999. **16**: p. 1641–1650.
132. Gouesbet, G., G. Grehan, and B. Maheu, *On the generalized lorenz-mie theory: first attempt to design a localized approximation to the computation of the coefficients g_nm.* J. Optics (Paris), 1989. **20**(1): p. 31-43.
133. Gouesbet, G., G. Grehan, and B. Maheu, *Computations of the g_n coefficients in*

the generalized Lorenz-Mie theory using three different methods. Appl. Opt. , 1988. **27**: p. 4874-4833.

134. Ren, K.F., G. Grehan, and G. Gouesbet, *Localized approximation of generalized Lorenz–Mie theory: Faster algorithm for computations of beam shape coefficients, g_nm.* Particle & Particle Syst. Characteriz, 1992. **9**: p. 144–150.
135. Gouesbet, G., G. Grehan, and B. Maheu, *Localized interpretation to compute all the coefficients g_nm in the generalized Lorenz-Mie theory.* J. Opt. Soc. Am. A, 1990. **7**(6): p. 998-1007.
136. Gouesbet, G. and J.A. Lock, *A rigorous justification of the localized approximation to the beam-shape coefficients in the generalized Lorenz–Mie theory. II: Off-axis beams.* J. Opt. Soc. Amer. A,, 1994. **11**: p. 2516–2525.
137. Grehan, G., F. Guilloteau, and G. Gouesbet, *Optical levitation experiments for generalized Lorenz–Mie Theory Validation.* Particle & Particle Syst. Characteriz., 1990. **7**: p. 248–249.
138. Ren, K.F., G. Grehan, and G. Gouesbet, *Radiation pressure forces exerted on a particle arbitrarily located in a Gaussian beam by using the generalized Lorenz-Mie theory, and associated resonance effects.* Optics Communications, 1994. **108** p. 343-354.
139. Ren, K.F., G. Grehan, and G. Gouesbet, *Prediction of reverse radiation pressure by generalized Lorenz–Mie theory.* Applied Optics, 1996. **35**(15): p. 2702-2710.
140. Martinot-Lagarde, G., B. Pouligny, M.I. Angelova, G. Grehan, and G. Gouesbet, *Trapping and levitation of a dielectric sphere with off-centered Gaussian beams: II. GLMT analysis.* Pure & Appl. Opt.: J. Eur. Opt. Soc. Pt. A, 1995. **4**: p. 571-585.
141. Jones, F.E., *Evaporation of Water: With Emphasis on Applications and Measurements*. 1991: CRC-Press.
142. Fuchs, N.A., *Evaporation and Droplet Growth in Gaseous Media*, ed. R.S. Bradley. 1959, New York: Pergamon Press, Inc.
143. Kinzer, G.D. and R. Gunn, *The Evapoaration, Temperature and Thermal Relaxation-Time of Freely Falling Water Drops in Stagnant Air.* J. Meteorol., 1949. **6**: p. 243.
144. Duguid, H.A. and J.F. Stampfer Jr, *The Evaporation Rates of Small, Freely*

Falling Water Drops. Journal of the atmospheric sciences, 1971. **28**: p. 1233-1243.

145. Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C*. 1993: Cambridge University Press, Cambridge, UK.
146. Weast, R.C., *CRC Handbook of Chemistry and Physics*, ed. Weast. 1997, Cleveland, OH: CRC Press. INC.
147. Chaplin, M. *Water Structure and Science*. [cited; Available from: http://www.lsbu.ac.uk/water/vibrat.html.
148. Prahl, S. *Optical Absorption of Water*. [cited; Available from: http://omlc.ogi.edu/spectra/water/index.html.
149. Warren, S.G., *Optical-constants of ice from the ultraviolet to the microwave.* Appl. Opt., 1984. **23** p. 1206-1225.
150. Quickenden, T.I. and J.A. Irvin, J. Chem Phys., 1980. **72** p. 4416.
151. Buiteveld, H., J.M.H. Hakvoort, and M. Donze, SPIE Proceedings on Ocean Optics XII, 1994. **2258** p. 174.
152. Einstein, A., *Investigation of the theory of Brownian movement.* Ann. D. physik, 1903. **17**: p. 549.
153. Reif, F., *Fundamentals of statistical and thermal physics*. 1965: McGrawHill.
154. Ahmadi, G., *lecture note for ME437/537*, Clarkson University.
155. Abuzeid, S., A.A. Busnaina, and G. Ahmadi, *Wall deposition of aerosol particles in a turbulent channel flow.* J. Aerosol Sci., 1991. **22**(1): p. 43-62.
156. Chen, S., C.S. Cheung, C.K. Chan, and C. Zhu, *Numerical simulation of aerosol collection in filters with staggered parallel rectangular fibres.* Computational Mechanics 2002. **28**(152-161).
157. Our-k, H., G. Ahmadi, and J.B. McLaughlin, *particle deposition in a directly simulated turbulent channel flow.* Phys. Fluids A 1993. **5**(6).
158. Uhlenbeck, E.G. and S.L. Ornstein, *On the theory of the Brownian motion.* Physics Rev., 1930. **36**: p. 823-841.
159. Chandrasekhar, S., *Stochastic problems in physics and astronomy.* Reviews of Modern Physics 1943. **15**: p. 1-89.
160. Gupta, D. and M. Peters, *A Brownian dynamics simulation of aerosol deposition onto spherical collectors.* J. Colloid Interface Sci., 1985. **104**.

161. Ren, K.F., G. Grehan, and G. Gouesbet, *Localized Approximation of Generalized Lorenz-Mie Theory: Faster Algorithm for Computations of Beam Shape Coefficients, gnm.* Part. Part. Syst. Charaet, 1992. **9**: p. 144-150.

162. Kihara, H., *Stopped-Flow Apparatus For X-Ray Scattering and XAFS.* Journal of Synchrotron Radiation 1994. **1**: p. 74-77

163. BioCat-18. *How to Design and Perform a Solution X-Ray Scattering Experiment*. [cited; Available from: http://www.bio.aps.anl.gov/techniques/SAXS-HOWTO.html.

164. Kuwamoto, S., S. Akiyama, and T. Fujisawa, *Radiation damage to a protein solution, detected by synchrotron X-ray small-angle scattering: dose-related considerations and suppression by cryoprotectants.* J. Synchrotron Rad., 2004. **11**: p. 462-468.

Author's Biography

Seung Joong Kim was born in Seoul, Korea, on May 23rd in 1977. He graduated from Seoul Science High School in February 1996 and Seoul National University in February 2003 with a Bachelor of Science degree in physics (*cum laude*). From August 1999 to January 2002, he worked as a software engineer for ECO, Co. Ltd in Seoul, before relocating to Urbana-Champaign, Illinois to pursue graduate study in physics in July 2003. He completed a Master of Science in physics from the University of Illinois in August 2004. Following the completion of his Ph.D., Mr. Kim will join the Andrej Sali group in UCSF (University of California at San Francisco) and be working on predicting the structure of proteins, as a postdoctoral fellow.