X-ray Studies of Ions at Liquid-liquid Interfaces in Model Systems for Solvent Extraction

BY

ZHU LIANG
B.S. China University of Geosciences, Wuhan, 2012
M.S. University of Illinois at Chicago, 2015

THESIS

Submitted as the partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Physics
in the Graduate College of the
University of Illinois at Chicago, 2020
Chicago, Illinois

To my parents, who led me to the world of science and have been supporting me unconditionally through the course of my life.

To my advisor, Mark Schlossman, who excels in scientific research while staying humble in teaching, and who patiently guides me through this adventure.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES VI
LIST OF FIGURES VII
LIST OF ABBREVIATIONS XIII
SUMMARY XIV
1 INTRODUCTION 1
1.1 Nuclear Fuel Cycle and Solvent Extraction 1
1.2 Actinide Lanthanide Separation Process (ALSEP) 2
1.3 Ion Transfer Across the Interface 3
2 OVERVIEW OF X-RAY SCATTERING THEORY 5
2.1 X-ray reflectivity (XR) 5
2.1.1 X-ray Refractive Index and critical angle 5
2.1.2 X-ray Reflectivity from an Ideally Flat Interface 7
2.1.3 Reflectivity from a Graded Interface – The Parratt Method 9
2.1.4 Electron Density Profile of Multiple Interfaces 11
2.2 X-ray Fluorescence Near Total Reflection (XFNTR) 13
2.2.1 X-ray emission line 13
2.2.2 X-ray Fluorescence Near Total Reflection (XFNTR) 14
3 EXPERIMENTAL SETUPS AND DATA ANALYSIS 18
3.1 X-ray Liquid Surface Instrument and Sample Cell 18
3.1.1 X-ray Liquid Surface Reflectometer 18
3.1.2 Thermostat and solvent-extraction sample cell 21
3.2 Sample Preparation 22
3.3 Level and Flatten The Sample Interface. 24
3.4 Basic Alignment and $\boldsymbol{Qz}$ Offset 28
3.5 Data measurement and analysis 33
3.5.1 Measuring XR Data With MW_Xreader software 33
3.5.2 Analyzing XR Data With XR Analyzer Software 33
3.5.3 Extract Fluorescence Data With Jupyter Notebook 36
3.5.4 Analyze Fluorescence With XFNTR Software 39
4 NANOSCALE VIEW OF ASSISTED ION TRANSPORT ACROSS THE LIQUID-LIQUID INTERFACE 43
4.1 Introduction 43
4.2 Results 46
4.3 Discussion 53
4.3.1 Static Interfacial Structures. 53
4.3.2 Consequences for the Mechanism of Extraction. 54
4.4 Materials and Methods 59
4.4.1 Materials and Sample Cell. 59
4.4.2 Fraction Extracted. 59
4.4.3 X-Ray Reflectivity and XFNTR. 59
4.4.4 MD Simulations. 60
4.4.5 Acknowledgments 60
4.5 Cited Literature 61

4.6 Supporting Information ... 65
4.6.1 X-Ray Fluorescence near Total Reflection Analysis of Y Samples ... 65
4.6.2 Composition of the Head Group Region in the Inverted Bilayer ... 66
4.6.3 Analysis of Y Inverted Bilayer Structure ... 66
4.6.4 Simulation Results ... 70
4.6.5 Fraction of Protonated Head Groups in DHDP Monolayer ... 70
4.6.6 Literature Values for κ, λ, and C_{sp} ... 71
5 PROBING THE INTERFACIAL ION DISTRIBUTION IN ALSEP BACK-EXTRACTION USING X-RAY FLUORESCENCE ... 77
5.1 Abstract ... 77
5.2 Introduction ... 78
5.3 Materials and Methods ... 81
5.3.1 Solution Preparation ... 81
5.3.2 ICP Measurements ... 83
5.3.3 Measurement Methods ... 83
5.3.4 X-ray fluorescence Near Total Reflection (XFNTR) Technique ... 84
5.4 Results and Discussion ... 98
5.5 Conclusion ... 103
5.6 Cited Literature ... 104
6 CONCLUSIONS ... 110
7 CITED LITERATURE ... **ERROR! BOOKMARK NOT DEFINED.**
APPENDIX A ... 112
A.1 Code for extracting data points from a fluorescence spectrum ... 112
A.1.1 beamprofile.ipynb ... 112
A.1.2 mca_plot-automatic.ipynb ... 114
A.1.3 mca_profile.py ... 117
A.1.4 mca_routines.py ... 117
A.1.5 mca_plot_routines.py ... 119
A.1.6 fit_routine.py ... 122
A.2 Code for XFNTR software ... 124
A.2.1 main.py ... 124
A.2.2 mplwidget.py ... 125
A.2.3 fit_ref.py ... 125
A.2.4 flu_geometry_routines.py ... 130
A.2.5 flu_routines_new.py ... 131
A.2.6 mainwindow.py ... 138
A.2.7 flu_geometry_2.py ... 152
A.3 Python functions for other calculations ... 153
A.3.1 Critical Angle ... 154
A.3.2 Electron density for a solution with given mass density ... 154
A.3.3 Electron density for a solution with mass density not given ... 155
A.3.4 Electron density profile calculation ... 156
A.3.5 Calculate Fresnel Reflectivity ... 157
A.3.6 Contribution of given element to absorption coefficient for water solutioin ... 158
A.3.7 Interfacial roughness using Capillary Theory ... 158
A.3.8 Ratio of reflectivity to Fresnel reflectivity $\boldsymbol{R/RF}$ ... 159

APPENDIX B 161
APPENDIX C 165
C.1 Beam Profile measurement. Beamtime: 2019 July; beamline: APS15-IDC 165
C.2 Fits for CRL lens 167
APPENDIX D 170
D.1 Other important data for sample height scan on HDEHP sample 170
D.1.1 2019 December 170
D.2 Other important data for [HEH]EHP sample 179
D.2.1 2018 November run 179
D.2.2 2019 April run 181

LIST OF TABLES

TABLE PAGE

Table 1. Absorption coefficiens and electron densities for typical biphase systems investigated in this thesis. Beam energy at 20 keV (and 8.542keV for Eu emission line), temperature at 297 K.. 9

Table II. Energies and intensities of X-ray emission lines for the elements in this article........... 14

Table III Main parameters in xfntr software as seen in Chapter 5.. 41

Table IV. Best-fit parameters to the X-ray reflectivity data 52

Table V Bending rigidity κ from the literature 74

Table VI Line tension λ from the literature 75

Table 7. Typical sample compositions in the experiments 83

Table 8 Averaged X-ray results of ion distributions of four samples for three different aqueous composition: pure water, 1mM HNO3 and 0.5M Citric, compared with ICP-MS results (below X-ray restuls)... 99

LIST OF FIGURES

FIGURE PAGE

Figure 1. Molecular structure of important chemicals in this thesis. Top left: HEH[EHP]; top right: HDEHP; middle: DHDP, lower left: citric acid and lower right: DTPA...................................... 3

Figure 2. Left: Refraction and reflection at the interface; right: Fresnel reflectivity (y-axis) as a function of Qz (x-axis).. 6

Figure 3. (a) illustration of how electron density profile is divided into layers in Parratt method. (b) Illustration of the propagation of electromagnetic wave with each layer. 10

Figure 4. Schematic electron transition process for emission lines from each shell. 13

Figure 5. (a)-(c) X-ray transmissivity, reflectivity and penetration depth as a function of momentum transfer. (d)-(e) XFNTR data (dots) and analysis (line) in Qz mode for samples with metal ions at the interface, in aqueous phase and in organic phase. (d) shows the monolayer of DHDP complexed with Y^{3+} at the interface that produces the data shown in (a) from an interface between 10−4 M DHDP in dodecane and 3 × 10−7 M YCl3 in water (pH 2.5) at 28 °C. (e) shows the XFNTR data for 50mM $Eu(NO_3)_3$ in aqueous phase and pure dodecane in organic phase, (f) shows 10mM HDEHP loaded with 1mM Eu in organic phase and pure water in aqueous phase. 16

Figure 6. XFNTR data (dots) and analysis (line) in sample height mode for samples with metal ions in aqueous phase (left), in organic phase as well at at the interface (right). 17

Figure 7. Illustration of the liquid surface reflectometer installed at APS 15-ID-C operated by ChemMatCARS (top) and the beam reflection and displacement used to refine $\alpha i = 0$ direction of $ki\alpha i$ (bottom). 19

Figure 8. Illustration of sample cell. 21

Figure 9 Illustration of leveling the sample. 25

Figure 10. Top: illustration of a transmission scan for a flat interface and an interface that is high or low. Bottom: a screenshot of MW_XReader solfware showing an example of a series of transmission scans while adjusting the interface. Y-axis is the beam intensity recorded by Pilatus1M detector; x-axis is the sample height in mm. 26

Figure 11. (a) Sketch of sample height scan. (b) Illustration of the curved edge. (c) MW_Xreader software showing a real sample scan data. 27

Figure 12. The reflectivity data for the interface of a water-dodecane whose oil phase contains 58mM HEH[EHP] and 1mM Eu, and water phase contains 0.5M Citric acid pH=3; The solid line is fitted with a $\delta Qz = 4 \times 10 - 4\ \text{Å} - 1$. 29

Figure 13. Illustration of “oscan” in which the height and orientation of the outcoming arm pivots around the center of the sample. The plot of intensity vs *oh* also shows a plateau whose center indicates the real height of oh. 30

Figure 14. Dialogs for $L2$ (top) and $L3$ (bottom) calculation. 31

Figure 15. A screenshot for MW-XReader software. 33

Figure 16. The screenshot of MW_Xreader software 35

Figure 17. In the 2nd cell, choose the scan number you want to analyze. It should be a list containing at least one scan. For example: `scans = [312,315,318]`. 37

Figure 18. In the 3rd cell, choose which spectrum to plot. For example, `qz_indeces=[6,8,11]` will plot 7th, 9th, and 12th of all the spectra. `errorbar=True` will also plot error bars. You can also choose to plot all the spectra by setting `show_all=1`. Note that this option overrides your previous selection, so set `show_all=0` if you want to selectedly plot. 37

Figure 19. Example of fitting Eu $L\alpha$ emission line into a four-gaussian model. 38

Figure 20. Cell 6 serves to adjust the first column of data to be saved, according to the type of the scan. `type_str` defines the scan to be either a Qz-scan or an sh scan. If it is a Q scan, Qz values will be copied to the first column as it is. If it is an sh scan, the first column have to be changed to sample height range defined through `left`, `right` and number of `points` through the code above. 39

Figure 21. Dialogs for setting limits on varying parameters (left), and dialogs for setting varying steps for the parameter for which you want to calculate the error bar. 42

Figure 22. Plots for the scattering geometry created by "flu_geometry_2.py". x axis is the horizontal dimension and y axis is height enlarged by 1000 times. Critical information such as Qz, incident angle sh, curvature and footprint size are annotated to the lower right corner. 42

Figure 23 (A) The variation of X-ray reflectivity RðQz Þ with wave vector transfer Qz (perpendicular to the interface) normalized to the calculated Fresnel reflectivity RFðQzÞ, as measured from the interface between metal (Y, Er, Sr) chlorides in water (pH 2.5 for Y and Er, pH 5.3 for Sr) and 10−4 M DHDP in dodecane. Samples were prepared as described in the text and measured at a temperature a few degrees below each sample's adsorption transition To. The upper three curves were shifted for clarity, although R=RF → 1 as Qz → 0 for all measurements. Curves labeled ErHD and ErLD refer to high-density and low-density Er interfaces. Lines are the best fits to the model described in the text. (B) Electron density profiles determined by the fits in A, where the right three curves were shifted for clarity, although ρwater → 0.333e ·Å as z → − 20 Å for all curves before shifting. The profiles are rounded as the result of capillary wave roughness of the interface; the dashed line for Y shows an example of the underlying zero-roughness profile. (C) X-ray fluorescence near total internal reflection (XFNTR) data (dots) and analysis (line) from an interface between 10−4 M DHDP in dodecane and 3 × 10−7 M YCl3 in water (pH 2.5) at 28 °C.

Error bars (±1 SD) are generally smaller than or similar to the size of the dots in A and C. (D-F) Molecular representations of the interfacial structures with zero interfacial roughness. (D) Cartoon of the measured monolayer with Sr(II). (E) Cartoon of a hypothetical maximum density inverted bilayer. (F) Cartoon of the measured low-density (LD) inverted bilayer of DHDP with Er(III). High-density (HD) inverted bilayers containing Y(III) or Er(III) consist of an intermediate configuration of ion-extractant complexes to those shown in D and E, as described in the text. Red and blue boxes identify the ionextractant complexes; red indicates the "up" orientation, and blue is the "down" orientation. Panels E and F are modified and reprinted with permission from ref. 19 (Copyright 2014, American Chemical Society).. 47

Figure 24. MD simulation results. (A) Snapshot of inverted bilayer from the last frame of the simulation—water (Bottom, red and white), dodecane (Top, green), inverted bilayer: ions in blue, dodecane that started in the top leaflet is shown in green, dodecane that started in the bottom leaflet is colored cyan, DHDP molecules that started in the top leaflet are colored gold, and DHDP that started in the bottom leaflet is colored black. Topmost layer of dodecane is ordered at the vapor interface, which is not relevant for comparison with the results of X-ray measurements. A smaller, disordered layer of dodecane exists immediately adjacent to the inverted bilayer. (B) MD electron density profile averaged over the final 100 ns of the simulation: total density profile in solid green, water in black, dodecane in red, Er ions in blue, and DHDP in dashed green. (C) Er coordination showing only DHDP head groups and water molecules.. 51

Figure 25. Domain budding mechanism. (A) A flat region of bare interface (dodecane above, water below) becomes (B) spontaneously curved due to the adsorption of extractants and their interactions with ions (not shown) at the interface. (C) The reduction in length of the domain edge (dashed line) reduces the line tension energy, which balances the bending energy required to form a spherical reverse micelle. (D) Separation of the micelle from the interface extracts the ions (not shown) in the interior of the reverse micelle into the bulk organic phase. 57

Figure 26. Cartoon of the interaction of a bulky branched-chain extractant DEHP with (A) a divalent ion (and two DEHP molecules) and (B) a trivalent ion (and three DEHP molecules) at the liquid-liquid interface (represented by the line), which illustrates how the interaction with the ion produces a spontaneous curvature of the interface. 58

Figure 27 (A) Example of fluorescence data with the Y Kα X-ray emission line at ~14.9 keV. (B) Background fitting with an exponential decay function. (C) Peak fitting after background subtraction. (D) Fit with Gaussian peak and exponential background. Error bars represent ±1 SD. 72

Figure 28. Number of oxygen atoms within 3 Å of the Er ion. The red line represents a 100-ps sliding average. 72

Figure 29. Distribution of O–Er–O angles for the closest six O atoms surrounding each Er ion, averaged over the 100-ns length of the trajectory. Atoms surrounding each ion were identified by first calculating all Er–atom distances that are closer than 8 Å, and then selecting the closest six (which were always oxygen atoms). 73

Figure 30. Radial distribution functions g(r) between (Left) Er and water oxygen, (Right) Er and the four oxygen atoms bound to the DHDP phosphorus, where the black line represents the P–O oxygen, the blue line represents the P=O oxygen, and the red and green lines represent the two ester oxygen atoms.. 73

Figure 31. Radial distribution function g(r) between Er and the DHDP phosphorus 74

Figure 32. Molecular structure of (a) HDEHP and (b) citric acid. 82

Figure 33 (a) Schematic of liquid-liquid interface with incident X-ray beam. (b) Illustration of the form of the three terms in Equation (5.1) that represent fluorescence emitted by ions in three different regions: bulk dodecane phase , bulk water phase and the interface between them. 85

Figure 34. (a) Schematic diagram of X-ray paths used for the calculation of the fluorescence intensity. Red, brown and green rays strike the interfaces along the x-direction at values $x' < -l/2$, $-l/2 < x' < l/2$ and $x' > l/2$, respectively where x' indicates the position on the curved interface and the x-axis is positive to the left of center. (b) detailed view of the geometry around the position where red ray strikes the interface. Note that $x' < 0$ in this region. The incident angle and interfacial curvature are exaggerated for clarity. 88

Figure 35. XFNTR the citric acid sample containing 0.5 M citric acid and 0.1 mM DTPA in the aqueous phase and 1 mM Eu 10 mM HDEHP in the oil phase. (a) Qz-scan; (b) sh-scan at $Qz = 0.006\ \text{Å} - 1$; (c) sh-scan at $Qz = 0.015\ \text{Å} - 1$. Vertical lines mark values discussed in the text. 96

Figure 36 (a) the $L\alpha$ and $L\beta$ emission line of Europium and the fitted gaussian peaks. (b) Representative X-ray fluorescence near total reflection (XFNTR) data as a function of Qz from the liquid-liquid interface between an *n*-dodecane solution of 1 mM Eu with 10 mM HDEHP and different aqueous phases: pure water (red), 1mM HNO3 pH=3 solution (green) and 0.5M citric acid pH=3 solution (blue). 99

Figure 37. (a) Example of normalized distribution of excess ions with different Debye length; (b) Total fluorescence created by ions in the evanescent region. 103

LIST OF ABBREVIATIONS

ALSEP	Actinides-Lanthanides Separation Process
TALSPEAK	Trivalent Actinide Lanthanide Separation with Phosphorus- Reagent Extraction from Aqueous Complexes
XFNTR	X-ray Fluorescence Near Total Reflection
XR	X-ray Reflectivity
R/Rf	X-ray Reflectivity Normalized to Fresnel Reflectivity
MD	Molecular Dynamics
ED	Electron Density
HDEHP	Bis(2-ethylhexyl-phosphoric) acid
HEH[EHP]	2-ethylhexylphosphonic acid mono-2-ethylhexyl ester
DTPA	Diethylenetriamine pentaacetate
SC	Steering Crystal
AMW	Rinsing with Acetone, Methanol and Water in order

SUMMARY

This thesis describes X-ray studies of interfacial phenomenon in solvent extraction processes, specifically those using X-ray reflectivity and Fluorescence technique to investigate the metal ion distributions at the liquid-liquid interface. The research described herein consists of two separate sets of investigations: the first probes a model system for the forward extraction process and the second probes a model system for the back extraction process. Forward extraction involves the transport of ions from organic to aqueous phases and back extraction involves their transport in the opposite direction, from aqueous to organic phases. Back extraction is often referred to as stripping in the solvent extraction literature.

During solvent forward extraction, amphiphilic extractants assist the transport of metal ions across the liquid-liquid interface from an aqueous ionic solution to an organic solvent. Investigations of the role of the interface in ion transport challenge our ability to probe fast molecular processes at liquid-liquid interfaces on nanometer-length scales. Recent development of a thermal switch for solvent forward extraction has addressed this challenge, which has led to the characterization by X-ray surface scattering of interfacial intermediate states in the extraction process. Here, we review and extend these earlier results. We find that trivalent rare earth ions, Y(III) and Er(III), combine with di-hexadecyl phosphoric acid (DHDP) extractants to form inverted bilayer structures at the interface; these appear to be condensed phases of small ion-extractant complexes. The stability of this unconventional interfacial structure is verified by molecular dynamics simulations. The ion-extractant complexes at the interface are an intermediate state in the extraction process, characterizing the moment at which ions have been transported across the aqueous-organic interface, but have not yet been dispersed in the organic phase. In contrast, divalent Sr(II) forms an ion-extractant complex with DHDP that leaves it exposed to the water phase; this result implies that a second process that transports Sr(II) across the interface has yet to be observed. Calculations demonstrate that the budding of reverse micelles formed from interfacial Sr(II) ion-extractant complexes could transport Sr(II) across the interface. Our results suggest a connection between the observed interfacial structures and the extraction mechanism, which ultimately affects the extraction selectivity and kinetics.

Solvent extraction is a primary technology for separating actinide from lanthanide elements in the recycling of used nuclear fuel. During solvent extraction, the distribution of the solvent between two phases at equilibrium is determined by their relative solubility in those phases, while the kinetics of phase transfer is believed to be driven by liquid-liquid interfacial process. Despite the importance of understanding the kinetics of phase transfer and the corresponding role of the liquid-liquid interface, there have not been any prior studies of solvent back extraction that probe the liquid structure on the nanoscale. Numerous challenges are encountered in this sort of study, including those posed by demands of using synchrotron X-rays to probe a deeply buried liquid-liquid interface, the necessity to stabilize the ion distribution at the liquid-liquid interface, and the extraneous signals produced by ions in the neighboring bulk phases. Here we report the observation of the interfacial europium ion distribution in a model system for back extraction process that mimics chemical conditions within the Actinide-Lanthanide Separation Process (ALSEP). In addition, we report on advances in the X-ray technique used in this study, X-ray Fluorescence Near Total Reflection (XFNTR), which were required to complete the study. The system studied consists of the liquid-liquid interface between a dodecane solution and an aqueous solution. The dodecane solution was loaded with trivalent europium ions Eu(III) complexed with the extractant Bis(2-ethylhexyl-phosphoric) acid (HEH[EHP]). The aqueous solutions were either pure water or contained nitric acid or citric acid. The concentrations of europium in the bulk solutions was measured by XFNTR and ICP-MS, while analysis of the XFNTR data also measured the interfacial density of ions. These measurements detected Eu(III) ions at the interfaces with nitric and citric acid solution, though larger density of ions were measured when nitric acid solutions were used. These experimental results demonstrate the capability of XFNTR to quantitatively characterize the presence of ions at the liquid-liquid interface in the presence of ions in both adjoining bulk phases, which is a requirement for further *in situ* investigations of the role of the liquid-liquid interface in the ALSEP process.

1 INTRODUCTION

1.1 Nuclear Fuel Cycle and Solvent Extraction

The nuclear fuel cycle refers to the process of treating nuclear fuel from the initial stage of the mining of uranium, to its use in the production of energy, to its disposal as nuclear waste. Reprocessing the used fuel makes it available for a new round of nuclear reaction, hence closing the nuclear cycle.

During the nuclear reaction inside a reactor, the fissile isotopes in nuclear fuel are consumed and become fission products, most of which are radioactive waste. The nuclear cycle will eventually come to a stop as fissile isotopes are consumed and fission products produced, yielding spent nuclear fuel. For 3% low enriched uranium fuel, the spent fuel typically consists of roughly 1% ^{235}U, 95% ^{238}U, 1% Pu and 3% fission products. Spent fuel and other high-level radioactive waste are extremely hazardous. Safe management of these byproducts of nuclear power, including their storage and disposal continues to be a challenging problem.

In addition to disposal, spent fuel can be reprocessed for reuse through solvent extraction process, which focuses on the recovery of uranium, co-extraction of uranium and plutonium, minor actinides (Np, Am, Cm) and some fission products (e.g., Cs and Sr).

One of the well-known solvent extraction processes is the PUREX process (plutonium uranium reduction-extraction), which has been practiced at industrial scale in the nuclear industry for over six decades. Other methods to extract uranium from the dissolved used fuel solution includes the UREX process (Uranium Extraction), which uses acetohydroxamic acid (AHA) to reduce plutonium in the acidic feed solution to the trivalent state, and to form unextractable complexes of plutonium and neptunium, preventing tributylphosphate (TBP) from extracting them. This process is followed by the separation of transmuting americium, because of the chemical similarity of americium and curium, their separation from each other is a very difficult approach.[1] TALSPEAK, which is a robust one-step An/Ln separation process, is designed to solve this problem. It relies on the strong complexation of actinides and lanthanides by an organic extractant such as bis(2-ethylhexyl-phosphoric) acid (HDEHP), and the complexation of the actinides in the aqueous phase by a complexant such as diethylenetriamine pentaacetic acid (DTPA), in the presence of a buffer such as citric acid. [2]

1.2 Actinide Lanthanide Separation Process (ALSEP)

In addition to the advanced TALSPEAK process, researchers at Argonne National Laboratory (ANL) and Pacific Northwest National Laboratory (PNNL) have been investigating new solvent formulations that combine the functional steps achieved by the TRUEX and TALSPEAK processes into a single solvent extraction cycle.[3] This approach "has led to the development of two processes that involve combining a neutral extractant with an acidic extractant. The neutral extractant serves to co-extract the trivalent actinides and lanthanides from HNO_3 solutions (much like the TRUEX process), while the acidic extractant serves to hold the trivalent lanthanides in the organic phase while the actinides are selectively stripped (i.e., back extracted) into a carboxylate-buffered solution containing a polyaminocarboxyllate ligand (a so-called reverse TALSPEAK process). One such approach is the Actinide Lanthanide Separation process (ALSEP), which utilizes TODGA or TEHDGA as the neutral extractant and HEH[EHP] as the acidic extractant. This new acidic complexing agent varies from conventional HDEHP in that one of the diethylhexyl groups is bonded directly to the phosphorus atom, rather than through an ether oxygen."

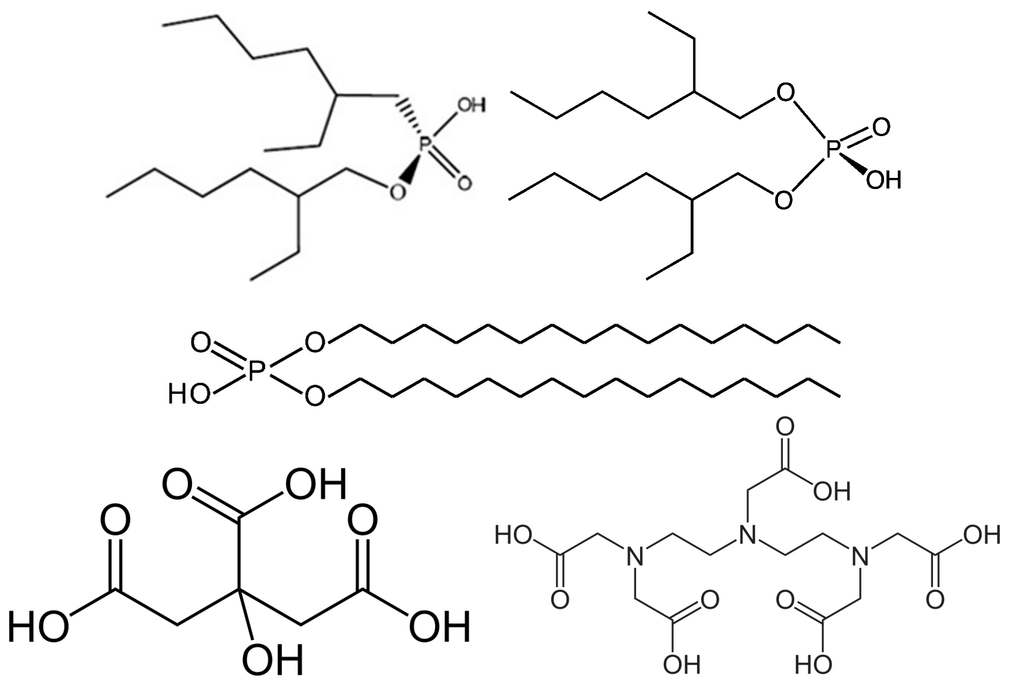


Figure 1. Molecular structure of important chemicals in this thesis. Top left: HEH[EHP]; top right: HDEHP; middle: DHDP, lower left: citric acid and lower right: DTPA.

1.3 Ion Transfer Across the Interface

The transfer of metal ions from aqueous to organic phase and back underlies the process of solvent extraction. It is assisted by the formation of supramolecular complexes with a soluble organic extractant.[4] Acidic organo-phosphorus extractants are used extensively and will be studied in this thesis. They are amphiphilic molecules with a phosphoric acid head group that binds to metal ions and hydrophobic alkyl tail groups that provide sufficient solubility in the organic phase. After extraction into the organic phase, metal ions are found in supramolecular ion–extractant complexes in the form of either coordination complexes or reverse micelles.[5–7] Although the interaction of metal ions with solutes at the organic–aqueous interface is likely to determine the efficiency and kinetics of extraction,[6] little is known about the mechanism of ion transport across this interface. Conventional hydrodynamic analysis assumes that ions diffuse across the interface on the nanoscale, although interfacial instabilities are predicted and observed on larger spatial scales.[7] In this thesis, we extend recent investigations of solvent extraction on the nanoscale to relate the observed interfacial intermediates to the extraction mechanism and efficiency, as well as suggest a role for nonequilibrium interfacial instabilities on the nanoscale.

2 OVERVIEW OF X-RAY SCATTERING THEORY

2.1 X-ray reflectivity (XR)

X-ray reflectivity (XR) probes the structure of interface by measuring the variation in electron density along the interfacial normal, the so-called electron density profile, with angstrom to sub-nanometer resolution. The electron density profile is the result of the arrangements of atoms and molecules at the interface. X-ray reflectivity is used to probe interfaces with both ordered and disordered arrangements of atoms and molecules. It is widely used in the fields of chemistry, physics and materials science to characterize surfaces and interfaces.

An X-ray beam is reflected from an interface and its intensity measured. If the interface is not well represented by a step function variation in the electron density along the interfacial normal, then the reflected intensity will deviate from that predicted by Fresnel reflectivity. The deviation can be analyzed to obtain the electron density profile of the interface.

The technique was first applied by Lyman G. Parratt in 1954,[8] whose initial work explored the surface of copper-coated glass, but since that time it has been extended to a wide range of both solid and liquid interfaces.

2.1.1 X-ray Refractive Index and critical angle

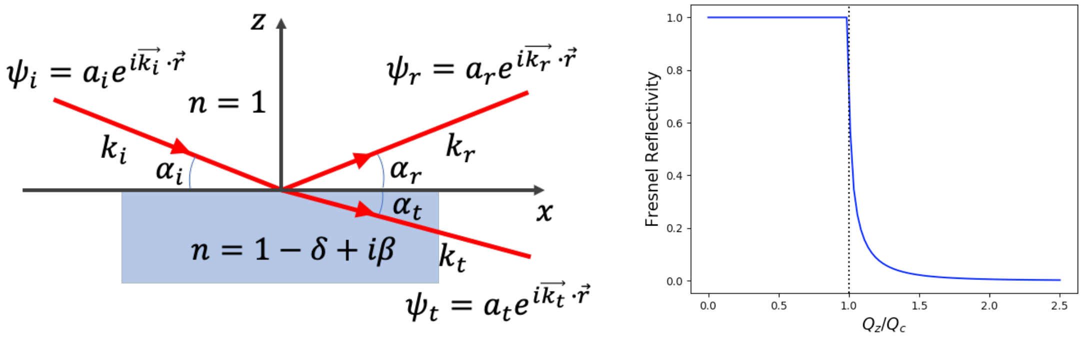
The simplest example of X-ray reflectivity is the reflection from a flat planer interface between vacuum and a single-component material. Assuming an X-ray beam of wave number $k_0 = 2\pi/\lambda$ in vacuum is reflected from such an interface, then the value of the index of refraction of n can be calculated by

$$\left|\overrightarrow{k_t}\right|/k_0 = n = 1 - \zeta + i\beta \tag{2.1}$$

where $|\overrightarrow{k_t}|$ is magnitude of the wavevector of the transmitted beam. The real and imaginary part of n account for the phase shift and attenuation of X-ray by the material and are given by

$$\zeta = 2\pi\rho_\infty r_e / k_0^2$$
$$\beta = \mu/2k_0 \tag{2.2}$$

where $r_e = e^2/(mc^2) \approx 2.818 \times 10^{-15}$ m is the classical radius of an electron, ρ_∞ is the average electron density of the bulk, and μ is the linear absorption coefficient for that particular material.



z
$\psi_i = a_i e^{i\overrightarrow{k_i}\cdot\vec{r}}$
$n = 1$
$\psi_r = a_r e^{i\overrightarrow{k_r}\cdot\vec{r}}$
k_i
α_i
α_r
k_r
α_t
x
$n = 1 - \delta + i\beta$
k_t
$\psi_t = a_t e^{i\overrightarrow{k_t}\cdot\vec{r}}$
Fresnel Reflectivity
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
Q_z/Q_c

Figure 2. Left: Refraction and reflection at the interface; right: Fresnel reflectivity (y-axis) as a function of Q_z (x-axis).

For common aqueous and organic solutions and a typical X-ray energy of ~20keV (wavelength $\lambda = 0.62$Å), $\zeta \approx 2 \times 10^{-6}$ and $\beta \approx 10^{-8}$. In light of the fact that $\beta \ll \zeta$, taking $\beta = 0$ is often a good approximation. Therefore, Snell's law corresponds to $\cos\alpha_i = n\cos\alpha_t$ where $n \approx 1 - \zeta$, and α_i and α_t are incident and transmitted angle, respectively. For small angles Snell's law can be rewritten as $\alpha_i^2 \approx \alpha_t^2 + 2\zeta$. Clearly, the beam is totally reflected with incident angle smaller than a threshold: $\alpha_i \leq \alpha_c$. This threshold α_c is critical angle and its expression is given by

$$\alpha_c \approx \sin^{-1}\left(\sqrt{2\zeta}\right) \approx \sqrt{2\zeta} \tag{2.3}$$

whose value is typically of the order of milli-radians for 20keV X-ray. Wavevector is a natural variable to describe X-ray scattering process. Here we define wavevector transfer as the difference between the wavevector of reflected and incident beam as

$$Q_z = |\overrightarrow{k_0} - \overrightarrow{k_r}| = 2k_0 \sin \alpha_i \tag{2.4}$$

Together with Eq. (2.2), The critical wave-vector transfer is thus given by

$$Q_c = 2k_0 \sin \alpha_c \approx 4\sqrt{\pi r_e \rho_\infty} \tag{2.5}$$

If the interface is between two different materials, rather than between one material and vacuum, then the critical angle is estimated with $\Delta\zeta = \zeta_t - \zeta_i$ instead of ζ where $\zeta_{i,t}$ refers to ζ for the materials containing the incident and transmitted beam. The critical wave-vector transfer is then given by

$$Q_c \approx 4\sqrt{\pi r_e \Delta\rho_\infty} \tag{2.6}$$

where $\rho_\infty = \rho_{\infty,t} - \rho_{\infty,i}$ is the difference of the electron density between two bulk materials. The typical value of Q_c for the water-dodecane interface falls in the range 0.010 ~ 0.011 Å^{-1} . Table 1 lists the electron densities and the critical angles for typical liquid-liquid interfaces.

2.1.2 X-ray Reflectivity from an Ideally Flat Interface

As shown in Figure 2, the incident wave vector is k_i, and the amplitude is a_i. Similarly, the reflected and transmitted wave vectors (at angle α_t) are k_r and k_t, respectively, and amplitudes are a_r and a_t. By imposing the boundary conditions that the wave and its derivatives at the interface $z = 0$, the amplitudes are related by

$$a_i + a_r = a_T \tag{2.7}$$

$$a_i k_i, + a_r k_r = a_t k_t \tag{2.8}$$

Taking the components of $\vec{k}$ parallel and perpendicular to the surface yields respectively

$$a_i k \; cos\, \alpha_i + a_r k \; cos\, \alpha_i = a_t (nk) \; sin\, \alpha_t \tag{2.9}$$

$$-(a_i - a_r) k \; sin\, \alpha_i = -a_t (nk) \; sin\, \alpha_t \tag{2.10}$$

from which one can readily derive Snell's law for an incident wave at α_i:

$$cos\,\alpha_i = n\,cos\,\alpha_t \tag{2.11}$$

and

$$\frac{a_i - a_r}{a_i + a_r} = n\frac{\sin\alpha_t}{\sin\alpha_i} \approx \frac{\alpha_t}{\alpha_i} \tag{2.12}$$

Plugging in equation (2.1) and as α_i and α_t are small, equation (2.11) can be expanded to yield

$$\begin{aligned}\alpha_i^2 &= \alpha_t^2 + 2\zeta - 2i\beta \\ &= \alpha_t^2 + \alpha_c^2 - 2i\beta\end{aligned} \tag{2.13}$$

If β is neglected, we can see that Fresnel equation can be derived from equation (2.12):

$$\begin{aligned} r &\equiv \frac{a_r}{a_i} = \frac{\alpha_i - \alpha_t}{\alpha_i + \alpha_t} = \frac{\alpha_i - (\alpha_i^2 - \alpha_c^2)^{1/2}}{\alpha_i + (\alpha_i^2 - \alpha_c^2)^{1/2}}; \\ t &\equiv \frac{a_t}{a_i} = \frac{2\alpha_i}{\alpha_i + \alpha_t} = \frac{2\alpha_i}{\alpha_i + (\alpha_i^2 - \alpha_c^2)^{1/2}}\end{aligned} \tag{2.14}$$

In the context of reflection and refraction, wavevector transfers are more useful than regular variables, one can rewrite equation (2.14) in terms of wavevector transfer:

$$\begin{aligned} R_F(Q_z) &= |r|^2 = \left|\frac{Q_z - \sqrt{Q_z^2 - Q_c^2}}{Q_z + \sqrt{Q_z^2 - Q_c^2}}\right|^2 \\ T(Q_z) &= |t|^2 = \left|\frac{2Q_z}{Q_z + \sqrt{Q_z^2 - Q_c^2}}\right|^2\end{aligned} \tag{2.15}$$

where subscript *F* represents Fresnel; $Q_z \equiv 2k\,sin\,\alpha_i \approx 2k\alpha_i$ and $Q_c \equiv 2k\,sin\,\alpha_c \approx 2k\alpha_c$ is the critical wavevector transfer derived in equation (2.6). Note that while the critical angle α_c depends on the X-ray wavelength, critical wavevector transfer Q_c is independent of it, $R_F(Q_z) = 1$ for $Q_z < Q_c$.

When $Q_z \ll Q_c$, total external reflection (i.e., $R_F(Q_z) = 1$) takes place, whereas for $Q_z \gg Q_c$, equation (2.15) reduces to

$$R_F(Q_z) \sim \left(\frac{Q_c}{2Q_z}\right)^4 \tag{2.16}$$

which indicates that Fresnel reflectivity falls off as $R_F(Q_z) \sim Q_z^{-4}$ (Figure 2).

Table 1. Absorption coefficiens and electron densities for typical biphase systems investigated in this thesis. Beam energy at 20 keV (and 8.542keV for Eu emission line), temperature at 297 K.

Oil phase	μ (cm^{-1})	$\rho_{\infty,i}$ (Å^{-3})	Aqueous phase	μ (cm^{-1})	$\rho_{\infty,t}$ (Å^{-3})	Q_c (Å^{-1})	α_c (mrad)
Dodecane and other dodecane solutions*	0.273 (7.451)	0.2591	Pure water and 1mM HNO_3	0.702 (26.58)	0.333	0.01023	0.505
			50mM $Eu(NO_3)_3$	1.015 (28.38)	0.335	0.01036	0.511
			0.5M citric acid	0.762 (28.68)	0.348	0.01122	0.553

* Other dodecane solutions include 10^{-4} DHDP solution, 10mM HDEHP/1mM Eu solution, they all have the same electron density and absorption coefficient as pure dodecane to the accuracy listed above.

Note that α_t is a complex number to be derived from equation (2.13) for a given incidence angle α_i. By decomposing α_t into its real and imaginary parts $\alpha_t = \mathrm{Re}(\alpha_t) + i\mathrm{Im}(\alpha_t)$, it can be seen that the transmitted wave falls off with increasing depth into the material as

$$a_t e^{i(k\alpha_t)z} = a_t e^{ik\mathrm{Re}(\alpha_t)z} e^{-k\mathrm{Im}(\alpha_t)z}$$

The intensity therefore falls off with a 1/e penetration depth Λ given by

$$\Lambda = \frac{1}{2k \cdot \mathrm{Im}(\alpha_t)} \tag{2.17}$$

2.1.3 Reflectivity from a Graded Interface – The Parratt Method

In real life experiments, the idealized interface is not realistic on the length scale probed by X-ray. In fact, many chemical and biological system form a buried layer between the two immiscible phases, in which case the average electron density varies along the surface normal at an atomic length scale comparable to the X-ray wavelength.

The solution to an arbitrary electron density profile $\rho(z)$ was first touched by Kiessig who explored the X-ray propagation in a stratified medium.[9] Later in 1954, Parratt developed a fully dynamical theory of X-ray propagation in a series of layers separating two infinite thick medium.[8]

The typical way of slicing the electron density profile $\rho(z)$ into J layers is shown in Figure 3(a). Layer j in (b) represent j^{th} layer in this scenario, bounded by the two interfaces at z_{j-1} and z_j, with a thickness of d_j. Layer $j = 0$ represents the semi-infinite layer of lower bulk and $j = J + 1$ the semi-infinite vacuum or upper bulk.

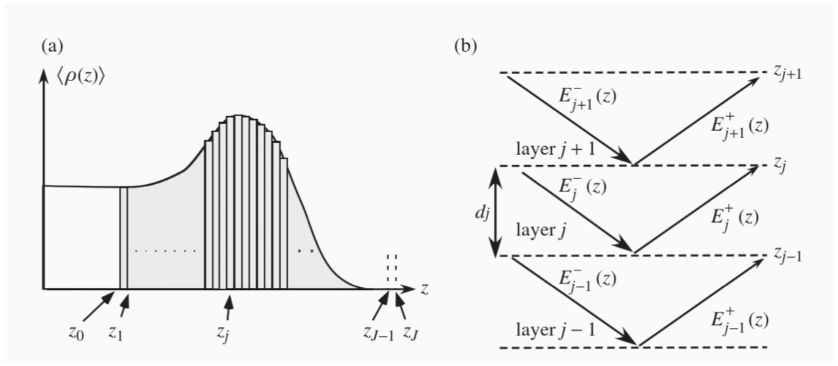


Figure 3. (a) illustration of how electron density profile is divided into layers in Parratt method. (b) Illustration of the propagation of electromagnetic wave with each layer.

In general, there are two plan waves propagating within the j^{th} layer, one propagates downward to the lower bulk denoted by $E_j^-(z)$, and one propagates upward to the top phase denoted by $E_j^+(z)$. The total field at position z are thus represented as a two-element matrix:

$$\boldsymbol{E}_j(z) = \begin{bmatrix} E_j^-(z) \\ E_j^+(z) \end{bmatrix} = \begin{bmatrix} A_j\, exp(-ik_{j,z}z) \\ B_j\, exp(ik_{j,z}z) \end{bmatrix} \tag{2.18}$$

where A_j and B_j are the magnitude of the plane wave $exp(\mp ik_{j,z}z)$. Specifically, $E_j(z_j)$ represents the total field in j^{th} layer at its upper boundary z_j, while $E_{j-1}(z_{j-1})$ represents the total field in the neighboring $(j-1)^{th}$ layer at its upper boundary z_{j-1}. The relation between these two waves, given by the following equation

$$E_{j-1}(z_{j-1}) = I_{j-1,j}E_j(z_{j-1}) = I_{j-1,j}\boldsymbol{P}_j\boldsymbol{E}_j(z_j) \tag{2.19}$$

works as a building block of the Parratt method. Here P_j is called the propagation matrix within layer j, and $I_{j-1,j}$ the interface matrix from layer j to layer $(j-1)$. They are giving by

$$P_j = \begin{bmatrix} exp(ik_{j,z}d_j) & 0 \\ 0 & exp(-ik_{j,z}d_j) \end{bmatrix}$$
$$I_{j-1,j} = \frac{1}{1+r_{j-1,j}}\begin{bmatrix} 1 & r_{j-1,j} \\ r_{j-1,j} & 1 \end{bmatrix} \tag{2.20}$$

Applying equation (2.19) successively on all the neighboring layers allows us to construct the relation of the electromagnetic fields between lower and upper bulk phase with

$$E_0(z_0) = ME_{J+1}(z_{J+1})$$

or

$$\begin{bmatrix} E_0^-(z_0) \\ E_0^+(z_0) \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}\begin{bmatrix} E_{J+1}^-(z_{J+1}) \\ E_{J+1}^+(z_{J+1}) \end{bmatrix} \tag{2.21}$$

where M is given by successive products of propagation matrix and interface matrix for each layer:

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = I_{0,1}P_1I_{1,2}P_2 \dots I_{J-1,J}P_JI_{J,J+1} \tag{2.22}$$

In view of the fact that the upward propagating wave in lower bulk phase is zero, i.e. $E_0^+(z_0) = 0$,the reflection coefficient and reflectivity are given by

$$r = E_{J+1}^+(z_{J+1})/E_{J+1}^-(z_{J+1})$$
$$R = |M_{21}/M_{22}|^2 \tag{2.23}$$

2.1.4 <u>Electron Density Profile of Multiple Interfaces</u>

Modeling the electron density profile have being widely investigated in the past century.[10] Consider the simplest example that constrains the electron density values far from the interface to be ρ_g and ρ_l, corresponding to the electron density of gas and liquid, respectively. The interfacial profile then can be modeled as

$$\rho(z) = \frac{1}{2}\left[\left(\rho_g + \rho_j\right) + \left(\rho_g - \rho_l\right)f(z)\right] \tag{2.24}$$

where f is a universal monotonic function such that $f(\pm\infty) = \pm 1$. The most common profile, first introduced by Buff, Lovett, and Stillinger,[11] is the error function:

$$f(z) = \operatorname{erf}\left(\frac{z}{\sqrt{2}\sigma}\right) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{z'^2}{2\sigma^2}\right] \mathrm{d}z' \tag{2.25}$$

where σ is the interfacial roughness, which can be calculated from capillary wave theory:[12]

$$\sigma^2 = \frac{k_B T}{2\pi\gamma} \ln \frac{4\sqrt{2}}{Q_z \Delta\beta r} \tag{2.26}$$

where k_B is the Boltzmann constant, T is the temperature in Kelvin, γ is the interfacial tension, and r is the average molecular radius. The angular acceptance of the detector is defined as $\pm\Delta\beta = h_3/2L_3$ where L_3 is the distance from sample to slit 3 (Figure 7) and h_3 is the vertical opening of slit 3.[10] For a sample whose aqueous phase is pure water and organic phase dodecane with DHDP, its roughness is 4.8 ± 0.1 Å. If a sample has metal ions, e.g. $ErBr_3$, in the water phase, the roughness of the interface can be as low as 3.8 ± 0.2 Å.[13]

In most cases of interfacial study, a single error function is insufficient to describe the interfacial structure between two phases. For example, the adsorption of DHDP onto the water-oil interface creates an electron density profile as a sum of multiple slabs. Equation (2.24) are then extended to be the following to account for multiple interface:

$$\rho(z) = \frac{1}{2}\sum_{i=0}^{N-1} f(z - z_i)(\rho_i - \rho_{i+1}) + \frac{\rho_0 + \rho_N}{2} \tag{2.27}$$

where $f(z - z_i)$ is the error function given in Eq. (2.23); N is the number of internal interfaces; ρ_i, ρ_0 and ρ_N is the electron density of slab I, the bulk aqueous and the bulk organic phase, respectively; z_i and $d_i = |z_i - z_{i-1}|$ are the vertical height and the thickness of each slab i, respectively. The calculation of X-ray reflectivity for a given electron density profile described in equation (2.27) is carried out with the Parratt method. Nonlinear least-square fitting is used to determine the minimum number, $N - 1$, required to fit the X-ray reflectivity data, as well as ρ_i and d_i for each slab. The data was normalized to the Fresnel reflectivity $R_F(Q_z)$ before the fitting.

2.2 X-ray Fluorescence Near Total Reflection (XFNTR)

2.2.1 X-ray emission line

When materials are exposed to X-rays with an energy greater than their ionization energy, one or more electrons from the inner orbital in the component atom may be kicked out, leaving a hole behind. The electronic structure of the atom then becomes unstable, and electrons in higher orbitals fall into the lower orbital to fill the hole left behind. In such process, energy is released in the form of a photon, the energy of which is equal to the energy difference of the two orbitals involved. Thus, the material emits radiation, or fluorescence, which has the energy characteristic of the atom present.[14]

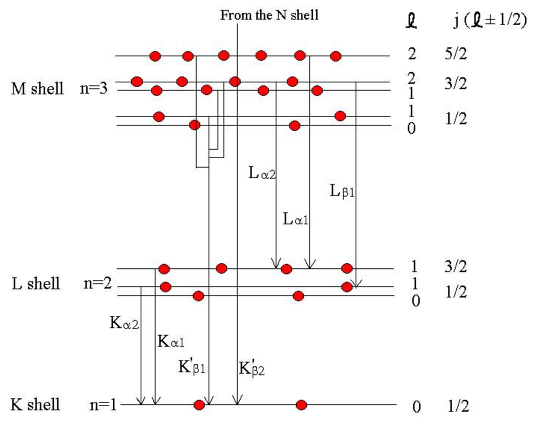


Figure 4. Schematic electron transition process for emission lines from each shell.

Table II listed the energies and intensities of the emission lines for Yttrium, Erbium, Strontium and Europium used in this research. The value 100 represents the maximum intensity an emission line can be for a specific element, while the absolute intensity varies between elements. Since each atom has electronic orbitals of characteristic energy, and there are a limited number of ways in which the fluorescence emission can happen, as shown in Figure 4. X-ray fluorescence is a useful technique to distinguish contributions from different types of atoms. In addition, the emission line intensity is directly related to the amount of ions/atoms present in the system, so it can also be used to find the concentration of a particular element in the system. In this thesis, the fluorescence technique is used to trace the Europium concentration around interface in a solvent back-extraction system.

Table II. Energies and intensities of X-ray emission lines for the elements in this article.

Element	Line	Energy(keV)	Relative intensity
Y	$K\alpha_1$	14.958	100
	$K\alpha_2$	14.883	52
Er	$L\alpha_1$	6.949	100
	$L\alpha_2$	6.905	11
Sr	$K\alpha_1$	14.165	100
	$K\alpha_2$	14.098	52
Eu	$L\alpha_1$	5.846	100
	$L\alpha_2$	5.817	11

2.2.2 X-ray Fluorescence Near Total Reflection (XFNTR)

XFNTR is a technique that measures the ion distribution for a given liquid-liquid sample using X-ray fluorescence. The method takes advantage of the dramatic change of three quantities — transmission coefficient, Fresnel reflectivity and penetration depth — below and above the critical angle. Those three quantities dominate the fluorescence signal produced by ions at the interface, in bulk oil phase and in bulk water phase, respectively. The detail of the XFNTR technique is described in Chapter 3.5.3 and Chapter 5.3.4.

An example of a system that contains metal ions at the interface is one that contains 3×10^{-7} M YCl_3 in water and 10^{-4} M DHDP in dodecane. The head groups of surfactants in the oil phase will complex with all metal ions and adhere to the interface, with metal ions merged in water phase. The fluorescence created by those metal ions are dominated by the transmission coefficient of the incident X-ray beam. The fluorescence data for such a sample as shown in Figure 5 (b) looks very similar to the plot of the transmission coefficient as a function of wavevector transfer derived from equation (2.15) for the same system.

The ion distribution in the organic phase is characterized by Fresnel reflectivity $R_F(Q_z)$ in equation (2.15). Figure 5 (d) shows a sample containing 10mM HDEHP loaded with 1mM Eu in its oil phase in contact with water. The fluorescence intensity produced by ions in organic phase has two origins: the incident beam and the reflected beam. While the intensity of incident beam is kept constant, the intensity of reflected beam follows Fresnel reflection, which equals to the incident intensity below critical angle and tumbles down to zero above critical angle as shown in Figure 5 (c). Therefore, the total fluorescence intensity features a 2:1 ratio below and above critical angle.

Furthermore, fluorescence intensity produced by ions in aqueous phase is not only affected by the transmission but also affected by the depth into the transmitted beam can reach into the material which raises from $60 \sim 80$Å below critical angle to $1 \sim 2\, \mu m$ around critical angle (Figure 5 (e)). An example is a sample containing 50mM $Eu(NO_3)_3$ in its aqueous phase in contact with pure dodecane as shown in Figure 5 (f).

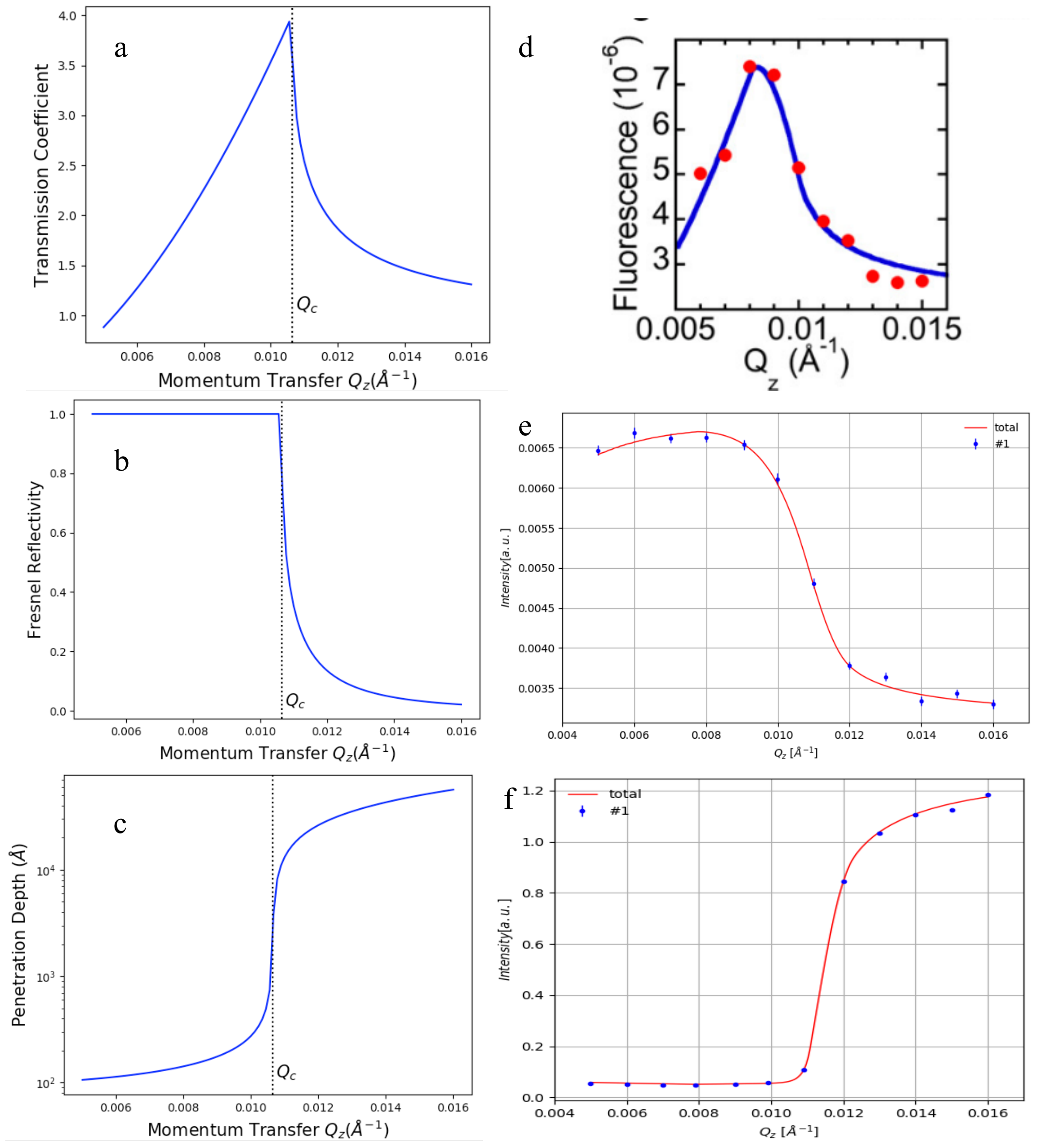


Figure 5. (a)-(c) X-ray transmissivity, reflectivity and penetration depth as a function of momentum transfer. (d)-(e) XFNTR data (dots) and analysis (line) in Q_z mode for samples with metal ions at the interface, in aqueous phase and in organic phase. (d) shows the monolayer of DHDP complexed with Y^{3+} at the interface that produces the data shown in (a) from an interface between 10−4 M DHDP in dodecane and 3 × 10−7 M YCl3 in water (pH 2.5) at 28 °C. (e) shows the XFNTR data for 50mM $Eu(NO_3)_3$ in aqueous phase and pure dodecane in organic phase, (f) shows 10mM HDEHP loaded with 1mM Eu in organic phase and pure water in aqueous phase.

Finally, the overall fluorescence intensity also relies on the geometry of the sample cell configuration, particularly on the position of the beam within the detection region. By changing the beam position, one can tune the weight of the fluorescence intensity from each phase. This can be done by changing the sample height, which will be addressed in detail in Chapter 5.

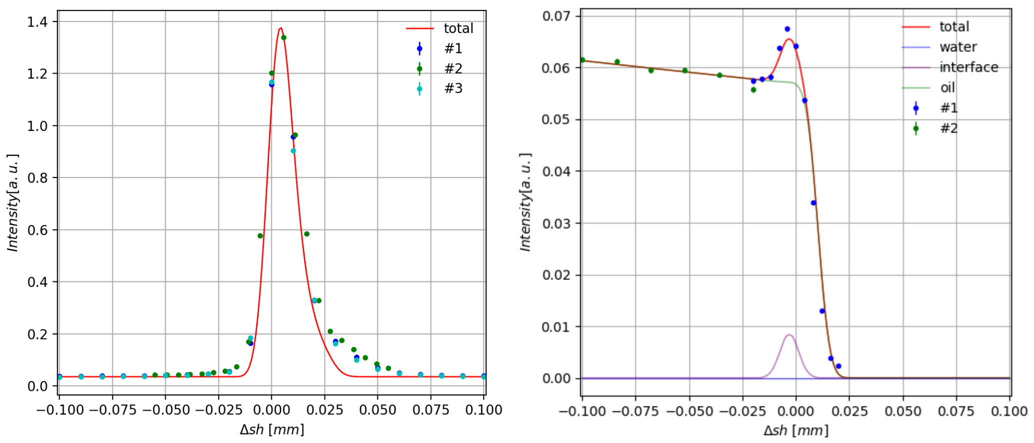


Figure 6. XFNTR data (dots) and analysis (line) in sample height mode for samples with metal ions in aqueous phase (left), in organic phase as well at at the interface (right).

3 EXPERIMENTAL SETUPS AND DATA ANALYSIS

3.1 X-ray Liquid Surface Instrument and Sample Cell

Two facts related to the X-ray scattering at the liquid-liquid interface set high standard for the instruments. One is that the weak interaction of X-ray with matter results very weak reflectivity signals. Only X-ray synchrotron sources are able to produce X-rays with high enough intensity. For example, the "third-generation" light sources produced at Advanced Photon Source (APS) at Argonne National Laboratory (ANL) provide high brilliance and highly monochromatic X-ray beams. All the experiments in this thesis are done at APS.

The other fact has to do with changing incident angles of X-ray shining striking the sample surface. For solid sample, it is relatively easy rotate the sample while X-ray beam is kept fixed. However, it is impossible to do so for liquid-liquid interface due to the nature of liquid surface requiring it to be horizontal all the time. The reflectometer for liquid surface experiment has the capability of changing the direction of X-ray beam while keeping the liquid sample still. The incoming X-ray beam is reflected from a so-called steering crystal so that its path is bended off by a small angle. Rotating the crystal in either parallel or perpendicular direction of the X-ray path effectively results in the change in the direction of X-ray beam, thus changing the incident angle with respect the horizontal plane.

3.1.1 X-ray Liquid Surface Reflectometer

All the experiments in this thesis use the reflectometer installed on the ChemMatCARS beam line at APS (15-ID-C), which is designed specifically to study horizontal liquid interfaces. An illustration of this reflectometer is shown in Figure 7.[10]

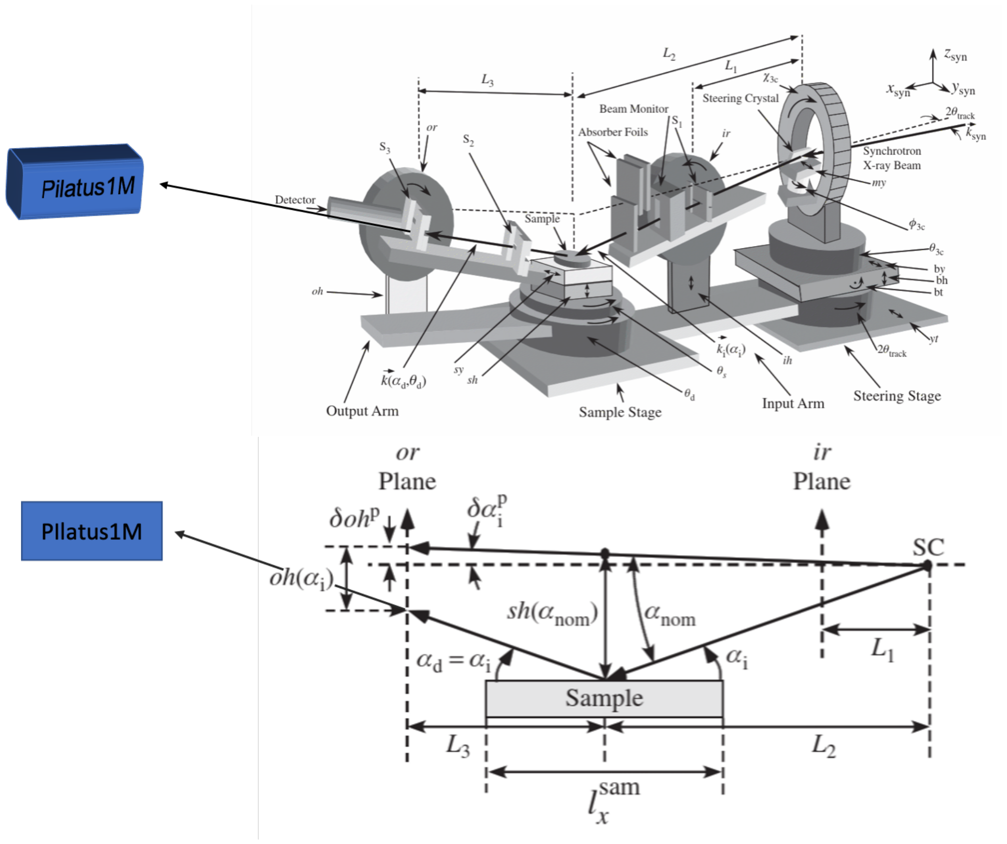


Figure 7. Illustration of the liquid surface reflectometer installed at APS 15-ID-C operated by ChemMatCARS (top) and the beam reflection and displacement used to refine $\alpha_i = 0$ direction of $\overrightarrow{k_i}(\alpha_i)$ (bottom).

Before the experiment starts, a series of careful alignments can place the beam and the sample at the right position, e.g. when incident angle $\alpha_i = 0$, beam points in horizontal direction and sample height $sh = 0$. However, these are just provisional position and more accurate measurements are needed to refine the zeros of α_i and sh. Assume that when beam is set horizontal by the instrument, the nominal direction (when the machine thinks the beam is horizontal, i.e. $\alpha_{nom} = 0$) deviates from the true horizontal by $\alpha_{real} = \delta\alpha_i^P > 0$ as sketched in Figure 7 (bottom). The point where the beam would intercept the or-plane would be higher than SC level (SC stands for steering crystal) by

$$\delta oh^P \approx (L_2 + L_3)\delta\alpha_i^P \tag{3.1}$$

Likewise, the vertical position at which the sample intercept the beam would be also be higher than SC level by an amount given by

$$\delta sh^P = L_2 \delta\alpha_i^P \tag{3.2}$$

In a condition where incident angle is nonzero, the instrument would bend down the beam by α_{nom} and lower the sample height by

$$sh(\alpha_{nom}) = L_2 \alpha_{nom} \tag{3.3}$$

in order for the beam to hit the interface, therefore the real sample height (with respect to SC level) is $sh(\alpha_{nom}) - \delta sh^P \approx L_2(\alpha_{nom} - \delta\alpha_i^P)$. Since the reflected angle is the same as the incident angle, the instrument would "think" that the reflected beam will intercept oh-plane at an angle α_{nom}, thus it will move oh position to

$$oh(\alpha_{nom}) \approx L_2\alpha_{nom} - L_3\alpha_{nom} \tag{3.4}$$

However, the true angle at which the beam strikes a flat horizontal surface is $\alpha_i = \alpha_{nom} - \delta\alpha_i^P$, and the true reflected angle $\alpha_d = \alpha_i$. The actual position of oh is lowered by

$$oh(\alpha_i) \approx L_2(\alpha_{nom} - \delta\alpha_i^P) - L_3(\alpha_{nom} - \delta\alpha_i^P) + \delta oh^P \tag{3.5}$$

From equation (3.4) and (3.5) we can calculate the offset in incident angle

$$\delta\alpha_i^P = [oh(\alpha_i) - oh(\alpha_{nom})]/(2L_3) \tag{3.6}$$

from which Q_z offset is obtained $Q_z^{off} = 2k_0\delta\alpha_i^P$. The deviation $\delta\alpha_i^P$ can be corrected by centering the beam on oh after it has been moved by

$$\delta oh^P \approx (L_2 + L_3)[oh(\alpha_i) - oh(\alpha_{nom})]/(2L_3). \tag{3.7}$$

The procedure used for correcting the $\delta\alpha_i^P$ error will reset all of the motors that are involved in tracking the incident arm, the sample position, and the detector.[10]

3.1.2 Thermostat and solvent-extraction sample cell

X-ray surface scattering in this thesis requires a very small incident angle (~5×10^{-4} rad), so it is critical for the interface to be flat. A good solution to this requirement is to increase the dimension of the interface along X-ray path. However, it will also increase the volume of the upper phase X-ray ray passes through, which will significantly increase the absorption of X-ray for liquid-liquid sample. A customized sample cell is designed to balance the two competing effects.

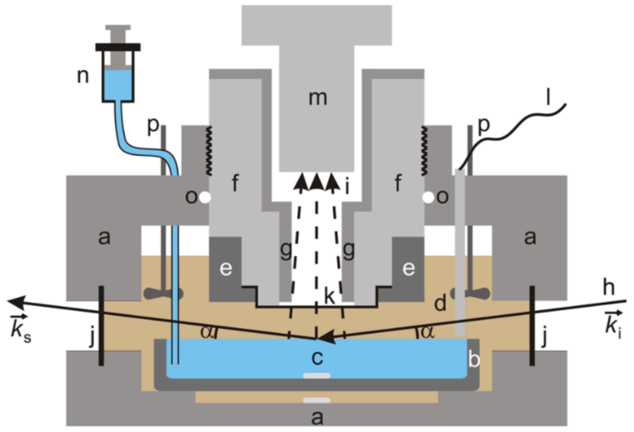


Figure 8. Illustration of sample cell.

As shown in Figure 8, the sample cell is made of Teflon-coated aluminum to increase the thermal conductivity (part *a*). A glass tray placed at the bottom of the sample cell holds the water phase, and the oil phase fills the rest of the space in the sample cell (part *b*-*d*). Two stirring motors are mounted on the lid so their propellers (part *p*) can stir the oil phase. A magnetic stir bar in the bottom of the glass tray serves to stir the water phase. The height of the glass tray is about the same as the center line of the Mylar window (part *j*) so that X-ray passing through the window reflects off the liquid-liquid interface. A tubing is inserted through the hole on the lid, one end touching the bottom of glass tray and the other end connected to a syringe (part *n*) containing the same aqueous solution. The whole sample cell sits on a tilting stage that is not drawn in the figure.

The interface can be flattened by adjusting the volume of the water phase and changing the tilting angle of the sample cell.

A stainless steel collimator is mounted at the center of the lid (part *e*-*g*), the bottom of which is a piece of Kapton film (part *k*). A vortex-60EX multi-cathode energy dispersive X-ray detector (SII Nano Technology USA, Inc., part *m*) is placed in the cylindrical well of the collimator. To minimize the absorption of X-ray by dodecane, the bottom of the well can be adjusted to as low as ~0.5mm above the interface without altering the interfacial shape.

3.2 Sample Preparation

All the aqueous and organic solutions are prepared in advance either at UIC or in the wet lab at APS sector 15ID.

At the beginning of each beamtime, the sample cell is thoroughly cleaned. Mylar window and o-rings are replaced with new ones. To disassemble the sample cell, all the screws on the side of the body are taken out and soaked in acetone for half an hour. Every piece of the sample cell should be rinsed in the order of acetone, methanol and water (AMW) before drying under the cover of Alphawipes (ITW Texwipe TX1009). The Kapton window at the bottom of the collimator is replaced with a 3" × 3" piece of Kapton film (7 μm), which is pre-cleaned with AMW. Note that this process is only required at the beginning of the beamtime, while for the following samples during the same beamtime, the body and the lid of the sample cell only need to be cleaned rinsing (AMW).

To reassemble the sample cell, the screws around the side window are tightened evenly to prevent leakage. After being put back together, the sample cell is filled with chloroform and sits for half an hour. Then the chloroform is replaced with Hexane and the sample cell is heated on a heater until the Hexane boils. To clean the sample lid with water, the Kapton window is rinsed for at least two minutes to minimize its chance to capture charges. Failing to do so might cause the Kapton film to catch charges that, when placed about 0.6 mm above the interface, will distort the interface and make it hard to flatten.

Every glass tray sits in the acid bath for at least half an hour before being placed in the sample cell. The acid bath is a beaker containing ~1600 mL sulfuric acid is used. For every 1000mL of sulfuric acid, there is 18.6g ammonium persulfate($(NH_4)_2S_2O_8$) added into the beaker to enhance the dehydrating power of the bath. The glass tray is rinsed with water after being taken out form the acid bath, then dried with high pressure nitrogen gas. The top of the glass tray is wetted all over its edges with a few drops of dodecane. This step allows the interface to catch the edge well thus makes it easy to flatten. Finally, a magnetic stir bar is placed in the glass tray before loading the glass tray into the sample cell.

When the sample cell with a glass tray placed inside is mounted to the sample stage, about 45 mL aqueous phase is immediately added into the glass tray with a 25 mL pipette. The water surface now should be slightly higher than then edge. The sample cell is then covered by Alphawipe and sits for 15 minutes for the dusts, if any, in aqueous phase to come to the interface. Next, the surface is aspirated by the tip of a 2mL Kimble disposable pipette connected to an aspirator, until the surface is about the same level of the glass edge. Flowing this step, 100mL oil phase is immediately loaded into the sample cell with a 50 mL pipette. After oil phase is added, the tubing the a syringe filled with the same aqueous solution is mounted onto the lid with its tip inserted all the way to the bottom of the aqueous phase. The syringe is then mounted to a Harvard syringe pump, which is used to add or remove as little as 10μL of liquid each time to fine adjust the height of the interface (see next section).

3.3 Level and Flatten The Sample Interface.

Most X-ray measurements are performed at an incident angle of a few miliradians. This is so small an angle that the footprint of a typical beam will span a few centimeters on the interface. For example, the X-ray beam produced at Advanced Photon Source 15-ID-C has a shape of a gaussian function whose full width half maximum is around 10μm. The minimum incident angle for an XFNTR scan is $Q_z = 0.005$ Å^{-1}, or $\alpha_i = 0.247$ mrad for 20keV. The footprint of the incident beam at a flat interface is 40.67 mm, which takes half of the interface. Therefore, the measured result is very sensitive to the flatness of the interface. However, the interface of a fresh made sample is usually far from being flat. It is crucial to flatten the interface before measurement. Usually it takes three steps to flatten the interface of a sample: visual leveling, transmission scan and sample height scan.

The first step is visual leveling. In this step the leveling of the sample cell is adjusted with the tilting stage, and the height of the interface adjusted with the syringe. As an example, the interface is lower than the glass edge as shown in Figure 9. One needs to make sure the sample is leveled, or placed horizontal, before adjusting the height of the interface. The distortion of the reflection renders a black stripe at the edge of the interface. That strip will vanish as one adjusts the tilting until the interface closer to the observer is leveled to the edge of the glass tray (Figure 9.b). The tilting stage now reads $t_1$for its current angular position. The same adjustment can be done for interface at the other edge of the glass tray (Figure 9.c) with reading t_2. The average of the two $(t_1 + t_2)/2$ is plainly the angular position for a leveled interface (Figure 9.a). The black strip now appears again since the interface is lower than the glass tray to begin with. One can pump in more liquid with the syringe until the black stripe vanishes.

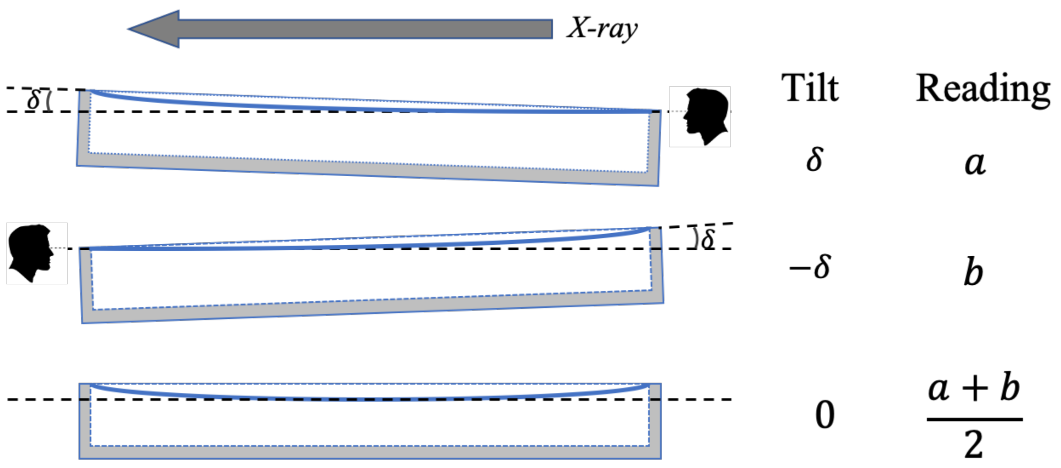


Figure 9 Illustration of leveling the sample.

Visual leveling flattens the interface coarsely with human efforts. In most cases, however, The flatness required by X-ray measurement cannot achieved by visual leveling only. One should rely on X-ray for further flattening, which involves two steps. The first step is the so-called transmission scan, whose intensity drop resolves the height of the interface at a much finer scale. In a transmission scan, X-ray is incident at $\alpha_i = 0$ as the sample moves from a lower position where the full beam penetrates through organic phase, all the way up until the glass tray completely blocks the beam. Figure 10 (top) shows the intensity recorded by Pilatus1M detector (Figure 7 top) drops from maximum to zero within a few dozen micrometers of sample height change when the beam starts to hit the sample. The height change for the intensity to vanish depends on the height of the liquid-liquid interface with respect to the glass tray edge. For a perfectly flat interface, the intensity drop is sharp and takes as little as 20μm (Figure 10 bottom, cyan), which is about the size of the beam. However, the interface of a newly made sample is usually low, and the intensity decreases slowly and spans ~50μm (Figure 10 bottom, blue). When this happens, one injectes 25 ~50 µL aqueous solution into aqueous phase and take a transmission scan to examine the height again. The adding of liquid followed by a transmission scan is repeated until the interface becomes flat and the intensity drop becomes sharp (Figure 10 bottom, cyan). Then the sample is moved to a position where the beam is roughly half-cut by the interface (point 9 on the cyan curve) and reset to zero.

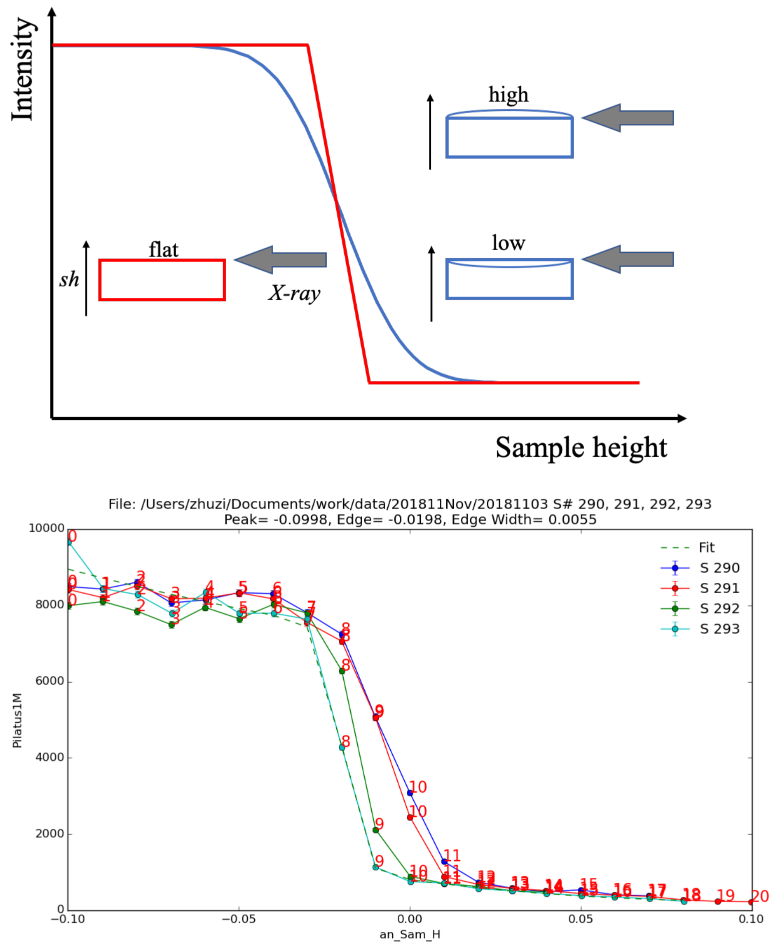
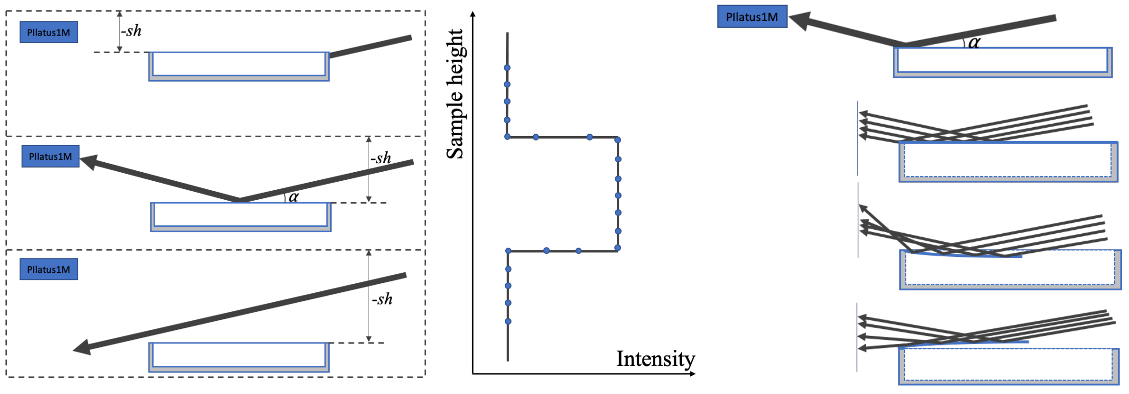
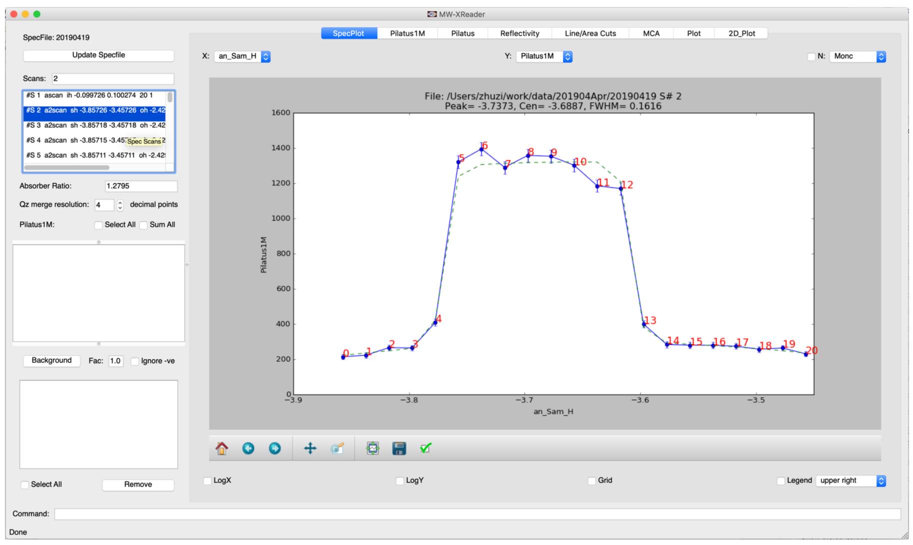


Figure 10. Top: illustration of a transmission scan for a flat interface and an interface that is high or low. Bottom: a screenshot of MW_XReader solfware showing an example of a series of transmission scans while adjusting the interface. Y-axis is the beam intensity recorded by Pilatus1M detector; x-axis is the sample height in mm.

The transmission scan is followed by a sample height scan for an even finer adjustment. The sample height scan is usually performed at $Q_z = 0.06$ Å^{-1}, or the incident angle α_i=2.96 mrad (20keV). Given the size of the beam (FWHM) which is ~ 10µm, the footprint of the beam on the interface is ~4 mm. Similar to transmission scan, the sample sets out at a lower position where the whole beam penetrates the oil phase, and moves up until the sample completely blocks the beam.

The motion is completed as a series of discrete steps Figure 11 (a). The reflected intensity at each step can thus be plotted as a function of sample height. Since only the reflected beam is recorded by the Pilatus1M detector, the plot features a plateau of non-zero intensity signal, whose width characterizes the flat area of the sample by $sh_{width} = d * \tan \alpha_i$ where d is the length of the flat part of the interface in the X-ray direction and α_i is the incident angle.





SpecFile: 20190419

Scans: 2

```
#S 1  ascan  ih -0.099726 0.100274  20 1
#S 2  a2scan  sh -3.85726 -3.45726  oh -2.42
#S 3  a2scan  sh -3.85718 -3.45718  oh -2.42
#S 4  a2scan  sh -3.85715 -3.45
#S 5  a2scan  sh -3.85711 -3.45711  oh -2.42
```

Absorber Ratio: 1.2795

Qz merge resolution: 4 decimal points

Pilatus1M: Select All Sum All

Fac: 1.0 Ignore -ve

X: an_Sam_H Y: Pilatus1M N: Monc

File: /Users/zhuzi/work/data/201904Apr/20190419 S# 2
Peak= -3.7373, Cen= -3.6887, FWHM= 0.1616

Legend: upper right

Command:

Done

Figure 11. (a) Sketch of sample height scan. (b) Illustration of the curved edge. (c) MW_Xreader software showing a real sample scan data.

Plateau width is very sensitive to, therefore a good indicator of, the flatness of the interface. Adding or removing as little as 10μm liquid from aqueous phase can lead to a change to the width. For a perfectly flat interface with $d = 75.6$ mm, which is the inner dimension of the glass tray, sh_{width} can be as large as 219 μm. Although in practice, 200 μm is a reasonable target width indicating that the interface is flat enough for X-ray measurement, a width smaller than 200 μm reveals no more information than that the interface is not flat. For example, from a sample height scan with a width of 160 μm one cannot tell whether the interface is high or low, or whether the sample cell is tilted in one direction or the other. A practical way is to discern that information from the CCD image of the reflection at the edge of the sample, or the left corner of the sample height scan. Figure 11 (b) shows the profile of the beam reflected from the edge of interface when the interface is flat, low and high. For the flat interface, the reflected beam has the same shape as the incident beam, so its CCD image is a dispersed flat dot. When the interface is lower than the edge, the reflection off the edge will be higher than that of the main beam, thus the CCD image shows an upward stroke from the main dot. Likewise, when the interface is higher than the edge, the reflection will be lower at the edge and the stroke in the CCD image will be downward. Figure 11 shows the reflection of the downstream side only, but one can readily derive that this effect is opposite for the upstream side. Combining this knowledge for both sample edge, one can tell the height and tilting of the interface, thus adjust it by adding/removing/tilting the sample as one does for visual leveling. Figure 11(c) shows a set of real sample height data loaded in the MW_Reader software.

3.4 Basic Alignment and Q_z Offset

As discussed in Chapter 3.1.1, there is a discrepancy between the set value of Q_z and the actual value. It is crucial that the difference, or $\delta Q_z = Q_z^{actual} - Q_z^{set}$, is measured and minimized before the measurement of sample. Failing to do so will introduce a large offset of Q_z in the data that will leave us uncertain about where the data point is in Q_z space.

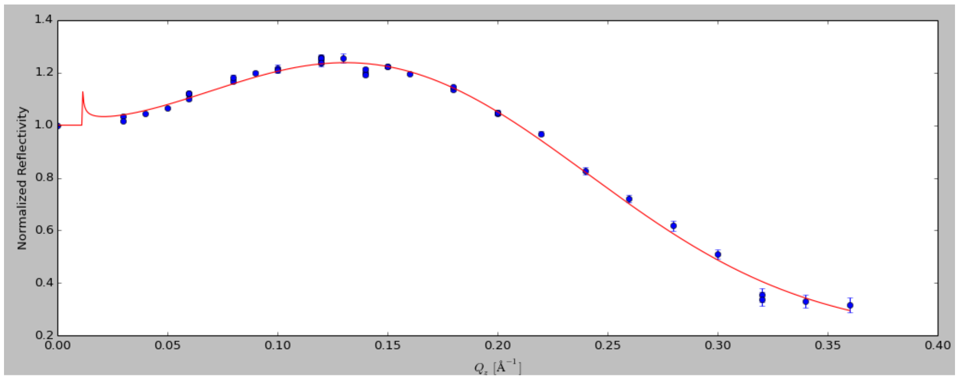


Figure 12. The reflectivity data for the interface of a water-dodecane whose oil phase contains 58mM HEH[EHP] and 1mM Eu, and water phase contains 0.5M Citric acid pH=3; The solid line is fitted with a $\delta Q_z = 4 \times 10^{-4}$ Å^{-1}.

The typical value of δQ_z are of the order of 10^{-4}, either negative or positive. Figure 12 shows a plot of R/R_f with a fitted $\delta Q_z = 6.0 \times 10^{-4}$ Å^{-1}, which causes the spike near zero. The practice of model optimization favors a fixed δQ_z because more fitting parameters will add to the complexity of the model. However, because the calculation of δQ_z is based on the measurements of many different variables, and each variable introduces an error bar that propagates and magnifies through the chain of calculation, the measured δQ_z ends up bearing a big uncertainty thus the benefit of having it fixed becomes very limited. Moreover, the reflectivity data on HEH[EHP] or HDEHP have few features as shown in Figure 12, therefore it is more dependent upon minimizing the number of fitting parameters.

A more practical way is to first minimize the measured δQ_z, then fit the δQ_z parameter as well as other key fitting parameters while analyzing reflectivity data. The advantage of doing this is one knows approximately what δQ_z should be, which places an upper limit for the fitted value of it. Chapter 3.1.1 elucidates the correction of $\delta \alpha_i^P$ hence the correction of δQ_z, and the correction relies on the measurement on L_2 and L_3 obtained by fitting the sample height and the height of the outcoming arm measured different incident angles to equation (3.3) and (3.4). Figure 14 shows an example of measuring L_2 and L_3 using "g_l2 & g_l3 calculation" dialog in the WM_Reader software (it can be found in the “Calculations” dropdown manu). The chosen angles are $Q_z = 0.06, 0.09, 0.12 \text{ and } 0.15 \text{ Å}^{-1}$, or $\alpha_{norm} = 2.96, 4.44, 5.92 \text{ and } 8.88 \text{ mrad}$, respectively. For each angle, a sample height scan and an “oscan” (explained below) are performed. The center position for the sample height scan is the real sample height $sh(\alpha_{nom})$, and all four values are typed in to the software (Figure 14, top) to fit to equation (3.3) from which L_2 can be obtained.

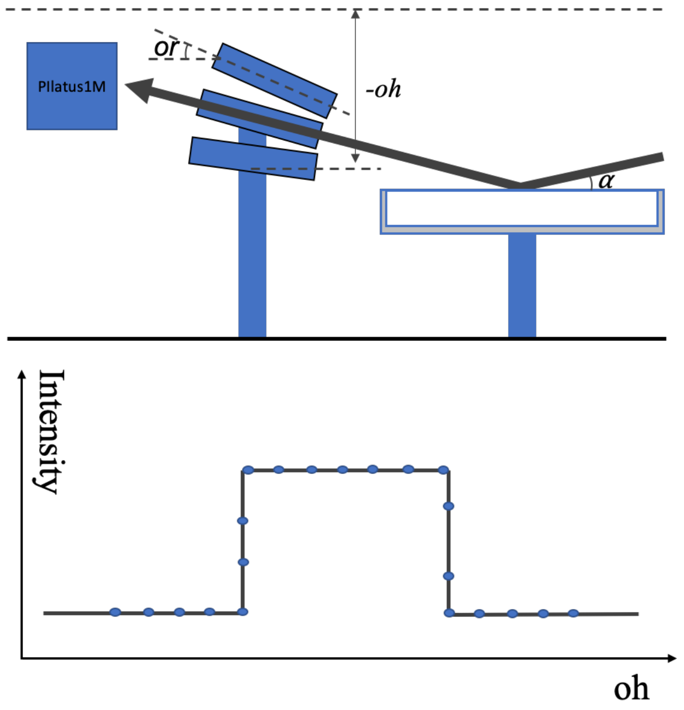
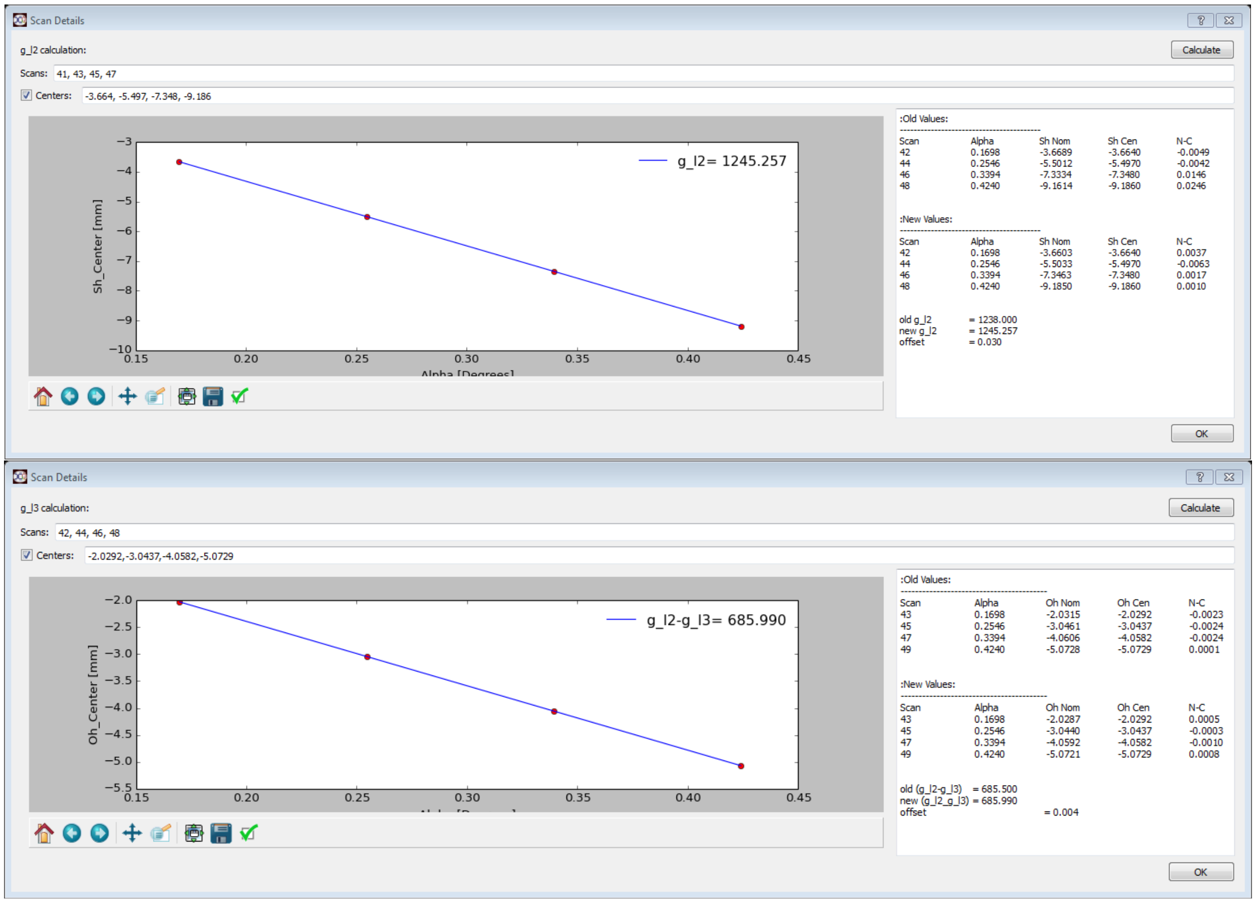


Figure 13. Illustration of “oscan” in which the height and orientation of the outcoming arm pivots around the center of the sample. The plot of intensity vs *oh* also shows a plateau whose center indicates the real height of *oh*.

The “oscan” is a combined scan by moving motor *oh,* which controls the height of the outcoming arm, and moter *or*, which controls the rotation of the outcoming arm, together so that the outcoming arm is always pointing toward the spot where the beam hit the interface. “oscan” requires the beam hit the center of the sample, therefore one should perform a sample height scan at the same incident angle first and relocate the sample at the center of the scan before moving forward to an “oscan”. Each sample height scan above is followed by an “oscan” (with the sample height centerd), whose center position is the real value of $oh(\alpha_{norm})$. All four center values are typed into the software (Figure 14 bottom) to fit to equation (3.4) from which $L_2 - L_3$ can be obtained. Note that $Q_z = 0.012$ Å^{-1}.

With newly measured L_2 and L_3 one readily calculates the offset $\delta\alpha^P$ hence Q_z^{off} using Eq.(3.6). However, setting L_2 and L_3 to the new values does not automatically eliminate $\delta\alpha^P$. One should go back to $Q_z = 0.006$ Å^{-1} and make sure the sample is centered to the sample height scan, then type "zero_angle" in the command line for final correction. The command calls a procedure that utilizes equation (3.7), and it involves resetting certain motors.[10] The procedure updates with a new smaller $\delta\alpha^P$, which can be calculated by repeating the steps above.



g_l2 calculation:

Scans: 41, 43, 45, 47

Centers: -3.664, -5.497, -7.348, -9.186

g_l2= 1245.257

```
:Old Values:
----------------------------------------
Scan    Alpha    Sh Nom    Sh Cen    N-C
42      0.1698   -3.6689   -3.6640   -0.0049
44      0.2546   -5.5012   -5.4970   -0.0042
46      0.3394   -7.3334   -7.3480   0.0146
48      0.4240   -9.1614   -9.1860   0.0246

:New Values:
----------------------------------------
Scan    Alpha    Sh Nom    Sh Cen    N-C
42      0.1698   -3.6603   -3.6640   0.0037
44      0.2546   -5.5033   -5.4970   -0.0063
46      0.3394   -7.3463   -7.3480   0.0017
48      0.4240   -9.1850   -9.1860   0.0010

old g_l2    = 1238.000
new g_l2    = 1245.257
offset      = 0.030
```

g_l3 calculation:

Scans: 42, 44, 46, 48

Centers: -2.0292,-3.0437,-4.0582,-5.0729

g_l2-g_l3= 685.990

```
:Old Values:
----------------------------------------
Scan    Alpha    Oh Nom    Oh Cen    N-C
43      0.1698   -2.0315   -2.0292   -0.0023
45      0.2546   -3.0461   -3.0437   -0.0024
47      0.3394   -4.0606   -4.0582   -0.0024
49      0.4240   -5.0728   -5.0729   0.0001

:New Values:
----------------------------------------
Scan    Alpha    Oh Nom    Oh Cen    N-C
43      0.1698   -2.0287   -2.0292   0.0005
45      0.2546   -3.0440   -3.0437   -0.0003
47      0.3394   -4.0592   -4.0582   -0.0010
49      0.4240   -5.0721   -5.0729   0.0008

old (g_l2-g_l3)   = 685.500
new (g_l2_g_l3) = 685.990
offset                    = 0.004
```

Figure 14. Dialogs for L_2 (top) and L_3 (bottom) calculation.

As shown in Figure 7 (bottom), L_2 and L_3 measure the distance from the SC to the sample center, and from the sample center to the rotating center of the outcoming arm, respectively.

Therefore, their values may vary according to the position of the sample. Whereas $L_2 + L_3$, which measures the distance from the SC to the rotating center of the outcoming arm, is independent of the position of the sample. However, the misalignment of the three-circle goniometer[10] may place the beam off the SC center and introduce an offset on $L_2 + L_3$ to begin with. In fact, the measured value of $L_2 + L_3$ from the above correction has a variation of up to 20 mm, indicating the beam might hit the SC somewhere rather than the center. Moreover, there is usually a clearance of ~3 mm for the sample to move freely on the sample stage, and the glass tray can also move freely by ~2mm inside a sample cell. As a result of all these freedoms as well as the fitting error, the uncertainty of both L_2 and L_3 can be up to 10mm. A typical value ranges from 1235 mm to 1245 mm for L_2 and 535 mm to 545mm for L_3.

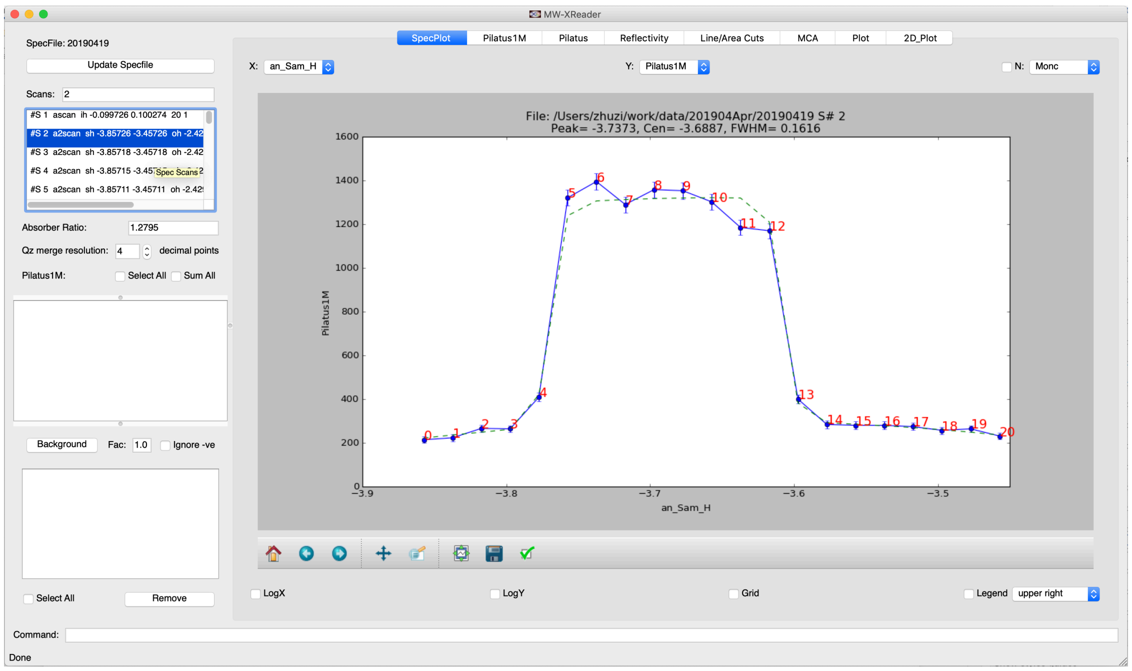
Correction to L_2 is also critical to XNFTR measurement too. The movement of the sample height is calculated using Eq.(3.2). If L_2 is different from its actual value by δL_2, sample height will be off by $\delta sh = -\delta L_2 \alpha_i$, and the footprint position where the beam strikes the interface will be off by δL_2 as well. However, the above correction bears too big an uncertainty of L_2 to precisely position the sample height. For an uncertainty of up to 10 mm in L_2, the uncertainty in the footprint position will be 10 mm, which is significant given the detector region is 12.7mm. One should do the correction within the Q_z range for the XFNTR measurement, or $0.006 \sim 0.016$ Å^{-1}. The suggested angle is $Q_z = 0.006, 0.009, 0.012$ and 0.015 Å^{-1}. In some cases, the critical angle falls very close to one of the chosen angles above. For example, the critical angle for the sample containing 0.5M citric acid in water and 10mM HDEHP in oil is 0.0112 Å^{-1}, which is very close to 0.012 Å^{-1}. One my substitute 0.013 Å^{-1} for it to stay away from critical angle. After the "zero angle" correction, one can check the alignment is by performing sample height scan at $Q_z =$

0.006 and 0.015. If the center of the scan is off the nominal position by less than 1 μm, the alignment is considered successful. Sketch

3.5 Data measurement and analysis

3.5.1 Measuring XR Data With MW_Xreader software

X-ray scattering measurement as well as sample alignment can produce huge amount of data, including up to a thousand CCD images, which makes it challenging to extract and visualize data by hand as the experiment goes. All the data used in thesis is taken with a python software called MW_XReader that can be obtained from the ChemMatCARS website.



SpecFile: 20190419

Scans: 2

```
#S 1  ascan  ih -0.099726 0.100274  20 1
#S 2  a2scan  sh -3.85726 -3.45726  oh -2.42
#S 3  a2scan  sh -3.85718 -3.45718  oh -2.42
#S 4  a2scan  sh -3.85715 -3.45
#S 5  a2scan  sh -3.85711 -3.45711  oh -2.42
```

Absorber Ratio: 1.2795

Qz merge resolution: 4 decimal points

Fac: 1.0

X: an_Sam_H Y: Pilatus1M N: Monc

File: /Users/zhuzi/work/data/201904Apr/20190419 S# 2
Peak= -3.7373, Cen= -3.6887, FWHM= 0.1616

Legend: upper right

Command:

Done

Figure 15. A screenshot for MW-XReader software.

3.5.2 Analyzing XR Data With XR Analyzer Software

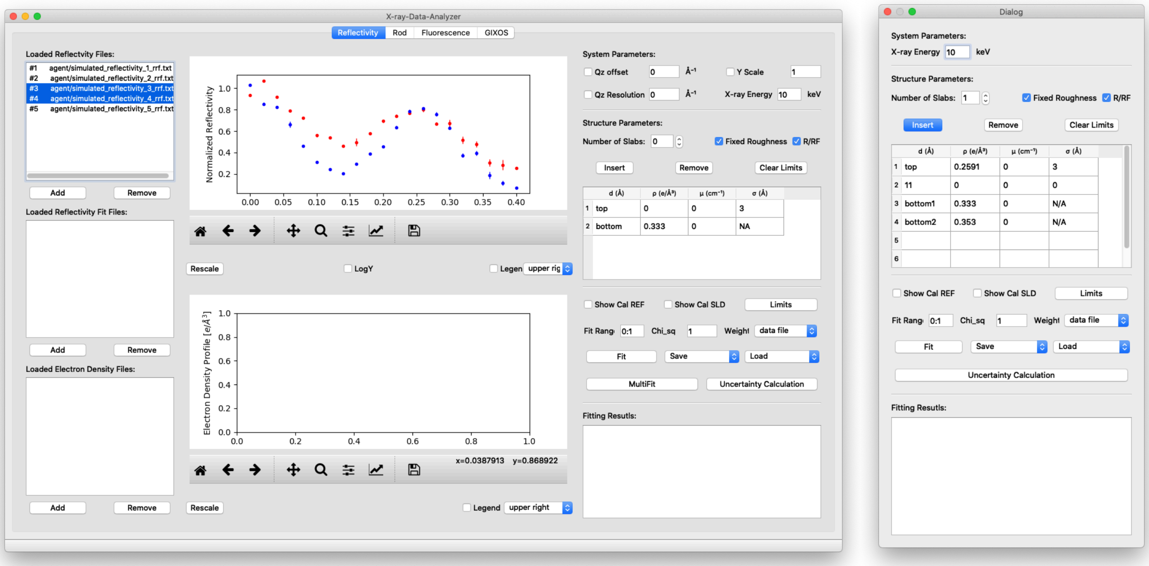
X-ray reflectivity analysis is carried out with the help of another python software: Xray_analyzer that can be obtained from the ChemMatCARS website.

The software has four main functions: Reflectivity, Rod, Fluorescence and GIXOS. Only Reflectivity is used in this thesis. Fluorescence part is migrated to a new software named "xfntr" that will be discussed in the following section.

The software is divided into three columns (Figure 16). The column on the far left serves as a space to load data to the software, the middle area serves to plot data, fitting results and electron density profile, while the right column is the parameter panel with which one can adjust fitting parameters.

The upper box titled "Loaded Reflectivity Files" in the left column displays data files to be fitted. Supported file format is a text file that contains 3 columns of values separated by whitespaces (either Tab or Space) representing $(x, y, yerr)$. The name of such a file should end with "_ref.txt" or "_rrf.txt", or the software won't recognize it. Imported data files appear in the box as selectable items, and one can select multiple files at once. The data of the selected files will appear as a series of dots in the upper plot area at the center of the software. The box in the middle titled "Loaded Reflectivity Fit Files" could also display additional sets of data imported by the user, as well as curves calculated from the fit parameters shown in the right most column. Supported file format is the same as that required for the upper box, except that the 3rd column, or *yerr*, is not required. The name of such a file ends with but not limited to "*_fit.txt". The data of the selected files in this box usually contains a lot more points, thus appear as solid lines in the plot, and they don't interfere with the fitting procedure. The lower box hosts electron density file which can be sent to the lower plot area as a solid line. They should have same file format as "*_fit.txt" files.

The panel on the right is a place to change parameters.



Loaded Reflectvity Files:
#1 agent/simulated_reflectivity_1_rrf.txt
#2 agent/simulated_reflectivity_2_rrf.txt
#3 agent/simulated_reflectivity_3_rrf.txt
#4 agent/simulated_reflectivity_4_rrf.txt
#5 agent/simulated_reflectivity_5_rrf.txt

Loaded Reflectivity Fit Files:

Loaded Electron Density Files:

x=0.0387913 y=0.868922

System Parameters:
Qz offset 0 Å⁻¹ Y Scale 1
Qz Resolution 0 Å⁻¹ X-ray Energy 10 keV

Structure Parameters:
Number of Slabs: 0 ✓ Fixed Roughness ✓ R/RF

	d (Å)	ρ (e/Å³)	μ (cm⁻¹)	σ (Å)
1	top	0	0	3
2	bottom	0.333	0	NA

Fit Rang 0:1 Chi_sq 1 Weight data file

Fitting Resutls:

System Parameters:
X-ray Energy 10 keV

Structure Parameters:
Number of Slabs: 1 ✓ Fixed Roughness ✓ R/RF

	d (Å)	ρ (e/Å³)	μ (cm⁻¹)	σ (Å)
1	top	0.2591	0	3
2	11	0	0	0
3	bottom1	0.333	0	N/A
4	bottom2	0.353	0	N/A
5				
6				

Fit Rang 0:1 Chi_sq 1 Weight data file

Fitting Resutls:

Figure 16. The screenshot of MW_Xreader software

Multi reflectivity fitting function can be invoked by clicking the “multifit” button on the right. The panel appears as a separate dialog which allows the fitting of up to 5 data sets simultaneously. This feature allows the interfacial electron density profile to be fitted more accurately by fitting up to 5 data sets simultaneously. The sample configurations that underly those data sets should share the same interfacial structure while having different electron density in aqueous phase (i.e. different ρ_0). For example, a liquid-liquid sample that has DHDPa adsorbed onto the interface will generate a characteristic R/R_f data revealing the height (d_i) and density (ρ_i) information of the DHDP layer at the interface. However, the d_i and ρ_i obtained from the fitting are often correlated and subject to big error bars.

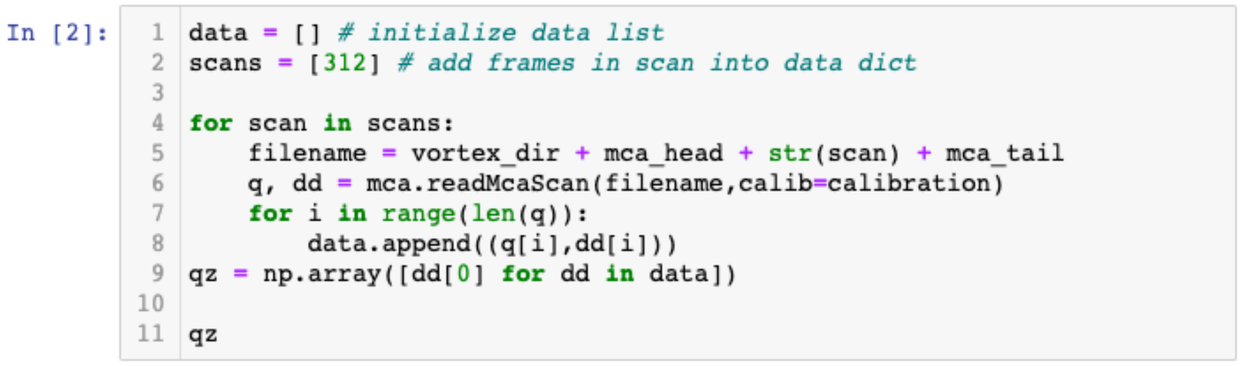
X-ray reflectivity method favors enhanced electron density contrast. The more the electron density of a slab differs from each other, the better R/R_f data resolves the height and density of each slab. One way to increase the electron density contrast between the two phase is to add Iohexol into the aqueous phase. Iohexol, or Omnipaque in trade name, contains Iodine and is a contrast agent used during X-rays.[15] The Iohexol is very chemically inert in aqueous phase, and its water mixture has an electron density ranging from 0.3 Å^{-3} (none in the water) to 0.43 Å^{-3} (0.74 M Iohexol in the water) Repeating X-ray measurement on samples containing same oil components but different fraction of Iohexol in water phase, one can obtain several data sets that are expected to yield same ρ_i, d_i and $\rho_N$with different ρ_0. Fitting those data all together will place a greater constraint on the parameter space, and eventually reduce the uncertainty of the fitting result. The operation on this panel is exactly the same as single-fitting panel, except that you need assign all the values of ρ_0 for the selected R/R_f data.

3.5.3 Extract Fluorescence Data With Jupyter Notebook

The immediate fluorescence data taken at APS 15-ID-C is the electronic response of the detector to the incoming X-ray over a spectrum of energy. If the element of interest is present in the system, the responses to its fluorescence will be strong at its emission energies. Figure 19 shows the response peaks for Europium at its $L\alpha$ (5.84 keV) and $L\beta$ (6.45 keV) emission energy. The $L\alpha$ peak is fitted to a Gaussian function, and the fluorescence intensity is calculated as the product of the height and FWHM of the Gaussian function.

This is a Jupyter notebook that works on extracting the intensity from raw spectrum data (Appendix A.1.2: “mca_plot-automatic.ipynb”). The “mca_profile.py” (see Appendix A) in a specific beamtime folder is configured to reflect that beamtime: “work_dir” for the absolute path of working directory; “vortex_dir” for the path where fluorescence raw spectrum is saved; “mca_head” (the prefix of the mca file name) for the name of the spec file for that beamtime. (e.g. “20191206_” for 2019Dec beamtime).

Once “mca_profile.py” is configured, one can go to “mca_plot-qutomatic.ipynb” and execute all the cells from the top down. A few modifications are explained below:

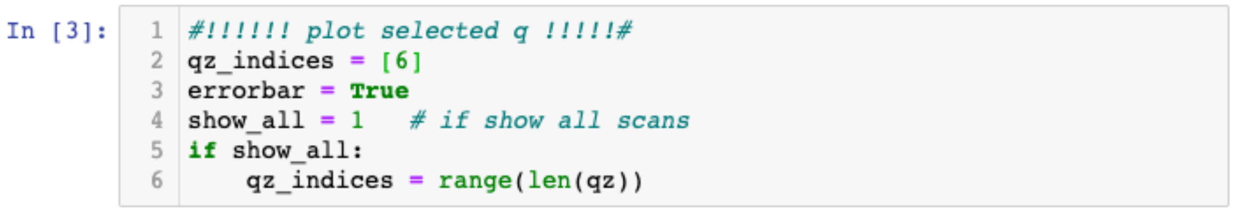


```
In [2]:
data = [] # initialize data list
scans = [312] # add frames in scan into data dict

for scan in scans:
    filename = vortex_dir + mca_head + str(scan) + mca_tail
    q, dd = mca.readMcaScan(filename,calib=calibration)
    for i in range(len(q)):
        data.append((q[i],dd[i]))
qz = np.array([dd[0] for dd in data])

qz
```

Figure 17. In the 2nd cell, choose the scan number you want to analyze. It should be a list containing at least one scan. For example: `scans = [312,315,318]`.



```
In [3]:
#!!!!!! plot selected q !!!!!#
qz_indices = [6]
errorbar = True
show_all = 1    # if show all scans
if show_all:
    qz_indices = range(len(qz))
```

Figure 18. In the 3rd cell, choose which spectrum to plot. For example, `qz_indeces=[6,8,11]` will plot 7th, 9th, and 12th of all the spectra. `errorbar=True` will also plot error bars. You can also choose to plot all the spectra by setting `show_all=1`. Note that this option overrides your previous selection, so set `show_all=0` if you want to selectedly plot.

The 4th cell and 5th cell plot out and fit the selected spectra, respectively. The 5th cell will also plot out all the fitted curve for each spectrum, and one can view the fitting result and decide whether to accept it. The cell implements a fitting procedure that can fit the data to a model containing up to 4 gaussian peaks with a background of either linear type or exponential type. Their parameters can be set in the `gauss_model` function in “fit_routine.py” (see Appendix A).

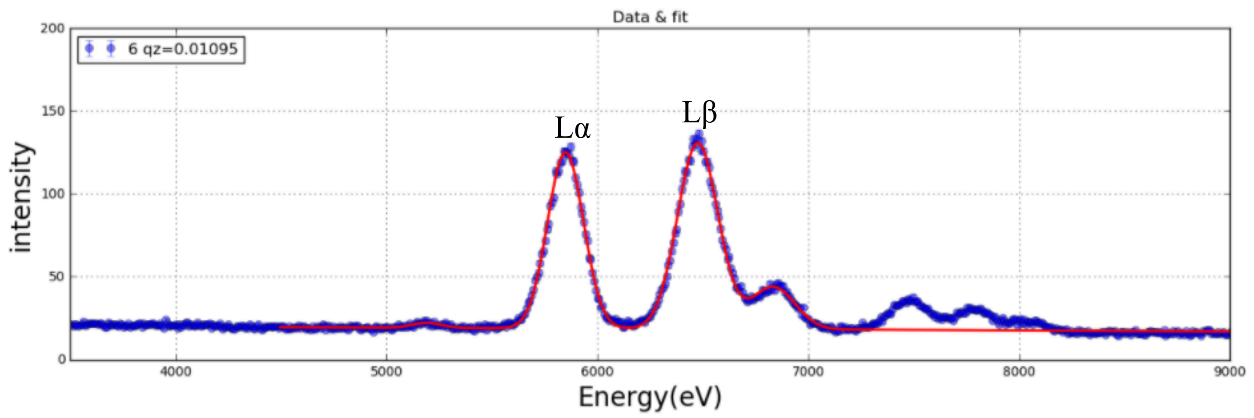
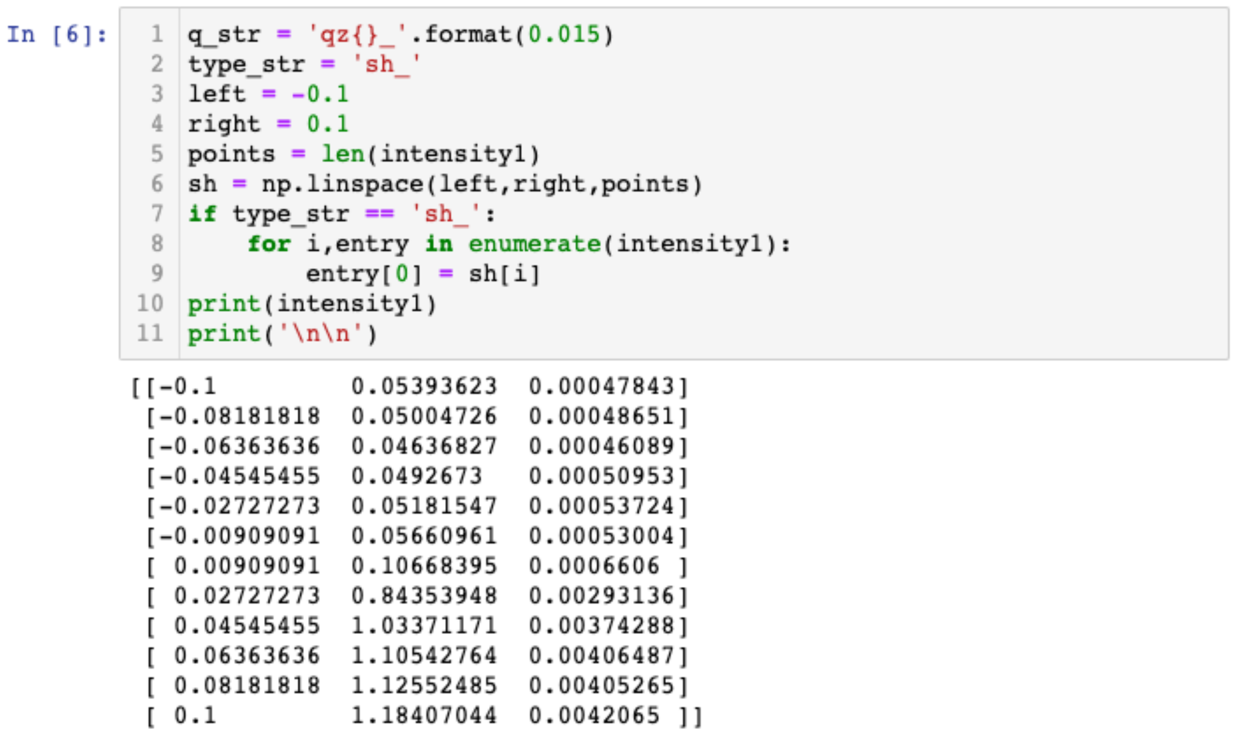


Figure 19. Example of fitting Eu $L\alpha$ emission line into a four-gaussian model.

One should pick a clean peak—a peak that contains only the element of interest—to be used for calculating the intensity. In the case shown in Figure 19, the $L\alpha$ peak instead of $L\beta$ peak is chosen because $L\beta$ peak is contaminated by iron $K\alpha$ peak at 6.4keV. One will need to fit the data to a multi-peak model whose positions can be set by changing `g_center` parameters in the `gauss_model` function in the "fit_routine.py" file. Other fine controls such as the range, height and width for gaussian peak, slope and offset for linear background, as well as amplitude and decay for exponential background can also be achieved by manually changing the code of the function.

The fitting procedure calculates the intensity as the multiplication of the fitted height and width of the emission peak (in this case the $L\alpha$ peak). Although the area calculated this way is not the true area, it is calculated consistently for all the data. Therefore, its ratio to the true area can counted in the calibration factor. Error bars are also calculated through error propagation. The same calculation is done for the spectrum at each Qz, and the result along with error bar are grouped as an entry and stacked as rows of a 2-D array, which can be exported to a text file. Once satisfied with the fitting result, one can proceed to cell 6 to confirm the type of the scan being analyzed as illustrated in Figure 20.



```
In [6]: q_str = 'qz{}_'.format(0.015)
        type_str = 'sh_'
        left = -0.1
        right = 0.1
        points = len(intensity1)
        sh = np.linspace(left,right,points)
        if type_str == 'sh_':
            for i,entry in enumerate(intensity1):
                entry[0] = sh[i]
        print(intensity1)
        print('\n\n')
```

```
[[-0.1         0.05393623  0.00047843]
 [-0.08181818  0.05004726  0.00048651]
 [-0.06363636  0.04636827  0.00046089]
 [-0.04545455  0.0492673   0.00050953]
 [-0.02727273  0.05181547  0.00053724]
 [-0.00909091  0.05660961  0.00053004]
 [ 0.00909091  0.10668395  0.0006606 ]
 [ 0.02727273  0.84353948  0.00293136]
 [ 0.04545455  1.03371171  0.00374288]
 [ 0.06363636  1.10542764  0.00406487]
 [ 0.08181818  1.12552485  0.00405265]
 [ 0.1         1.18407044  0.0042065 ]]
```

Figure 20. Cell 6 serves to adjust the first column of data to be saved, according to the type of the scan. `type_str` defines the scan to be either a Q_z-scan or an sh scan. If it is a Q scan, Q_z values will be copied to the first column as it is. If it is an sh scan, the first column have to be changed to sample height range defined through `left`, `right` and number of `points` through the code above.

The following cell plots out the final data as a function of Q_z or sh. And the data is ready to be saved at this stage.

3.5.4 Analyze Fluorescence With XFNTR Software

The software has three tabs in appearance: Data Extraction, Fluorescence and Geometry. Only the second tab is active and consists the main function for this software. The first tab—Data Extraction, which was covered in Chapter 3.5.3, and the third tab—Geometry, which will be explained later, will be implemented into this software in the future in a collaborative manner.

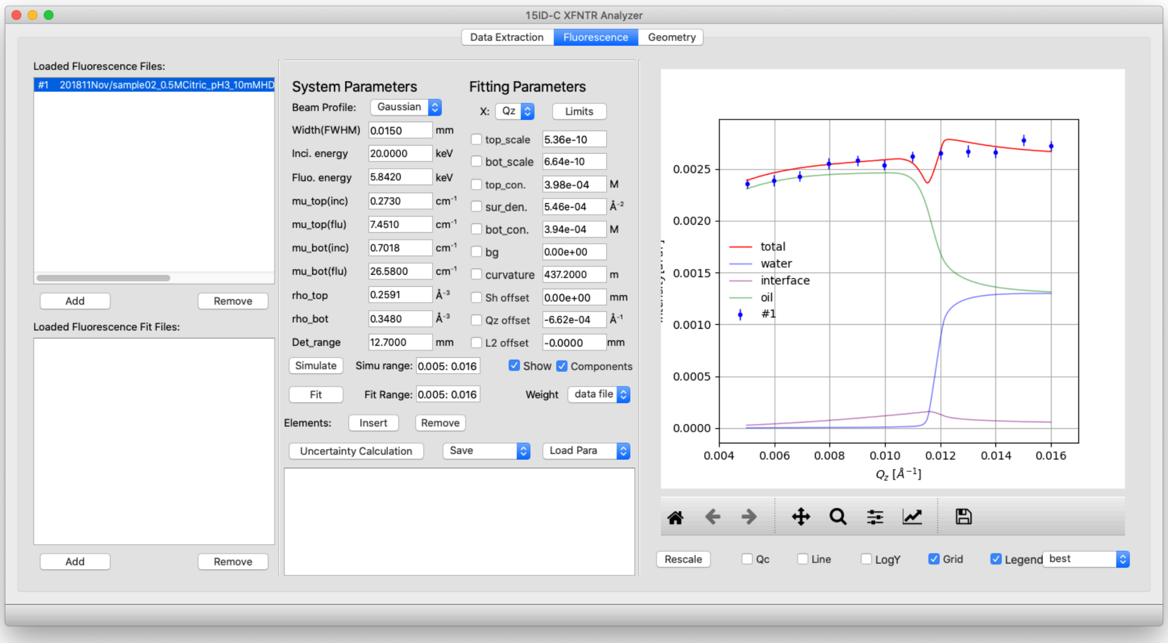
a. Installation

The software can be downloaded and installed through `pip install` command. First install anaconda and activate a python 3 environment, then type the command below:

```
pip install xfntr
```

and it will install the software as well as all its dependencies into the Anaconda environment. To execute the software, simply type `xfntr` in the command line while you are in your Anaconda python 3 environment.

b. *Fluorescence tab*



Loaded Fluorescence Files:
#1 201811Nov/sample02_0.5MCitric_pH3_10mMHD

Loaded Fluorescence Fit Files:

System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0150	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	$Å^{-3}$
rho_bot	0.3480	$Å^{-3}$
Det_range	12.7000	mm

Fitting Parameters

X: Qz

☐ top_scale	5.36e-10	
☐ bot_scale	6.64e-10	
☐ top_con.	3.98e-04	M
☐ sur_den.	5.46e-04	$Å^{-2}$
☐ bot_con.	3.94e-04	M
☐ bg	0.00e+00	
☐ curvature	437.2000	m
☐ Sh offset	0.00e+00	mm
☐ Qz offset	-6.62e-04	$Å^{-1}$
☐ L2 offset	-0.0000	mm

Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight data file

Elements:

☐ Qc ☐ Line ☐ LogY ✓ Grid ✓ Legend best

Similar to Data analysis software covered in Chapter 3.5.2. Again, it is divided into three columns, they are, from left to right, file, parameter and plot. The left most column hosts data and fit files as described previously.(see page 33); the middle column is the fitting panel which provides the values of all the parameters that characterize a fluorescence scan. The parameters fall into two categories. System parameters preset the sample and are known and fixed in the fitting. They include the energy and profile of the X-ray beam, its interaction with two bulk phases in terms of electron density, absorption coefficient, and the detection range of the fluorescence detector. Fitting parameters include all the unknown parameters to be optimized through the least-square fitting. Their corresponding symbols in the xfntr model described in Chapter 5 are listed in Table III.

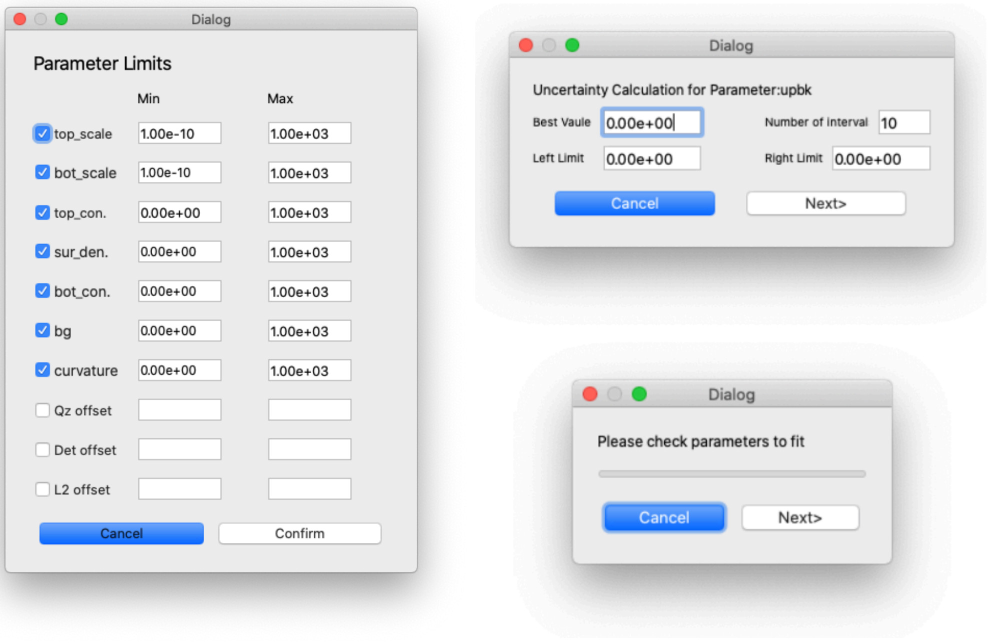
Table III Main parameters in xfntr software as seen in Chapter 5.

Name	Symbol	Name	Symbol	Name	Symbol
Width	$2.355\sigma_D$	mu_top(inc)	$\mu_{i,o}$	mu_top(flu)	$\mu_{e,o}$
Det_range	l	mu_bot(inc)	$\mu_{i,w}$	mu_bot(flu)	$\mu_{e,w}$
top_scale	C_1	bot_scale	C_2	top_con.	$N_A n_{oil}$
Bg	Constant	sur_den.	σ_{int}	bot_con.	$N_A n_w$
Curvature	R_C	Sh offset	Δsh	Qz offset	Δq_z
L2 offset	ΔL_2				

The first line in fitting parameters allows user to toggle between Q_z scan mode and sh scan mode. The simulation range and fit range along with "L2 offset" will change accordingly while all other parameters stay the same in each mode. The "Limits" button allows user to set lower and upper limit on selected parameters.

Every parameter is preceded by a checkbox which, if checked, marks the parameter as being fitted. Clicking "Fit" button will fit all the checked parameters to the data file highlighted in the "Loaded Fluorescence Files" to the left. Be aware that only one file can be selected while fitting. Clicking the "Simulate" button will calculate the theoretical fluorescence curve using the current values of all parameters and send the result to the plot area to the right if the "show" checkbox is checked.

If "Uncertainty Calculation" button is clicked, a dialog will pop up for users to set a sequence of values for the parameter to be calculate. Parameter of interest will be fixed at those values while all other parameters are fitted to minimum χ^2. A horizontal bar representing the target χ^2 value is plotted whose intersection with the fitted χ^2 curve will determine the left and right error bar for that parameter.



Dialog

Parameter Limits

	Min	Max
✓ top_scale	1.00e-10	1.00e+03
✓ bot_scale	1.00e-10	1.00e+03
✓ top_con.	0.00e+00	1.00e+03
✓ sur_den.	0.00e+00	1.00e+03
✓ bot_con.	0.00e+00	1.00e+03
✓ bg	0.00e+00	1.00e+03
✓ curvature	0.00e+00	1.00e+03
☐ Qz offset		
☐ Det offset		
☐ L2 offset		

Dialog

Uncertainty Calculation for Parameter:upbk

Best Vaule	0.00e+00	Number of interval	10
Left Limit	0.00e+00	Right Limit	0.00e+00

Dialog

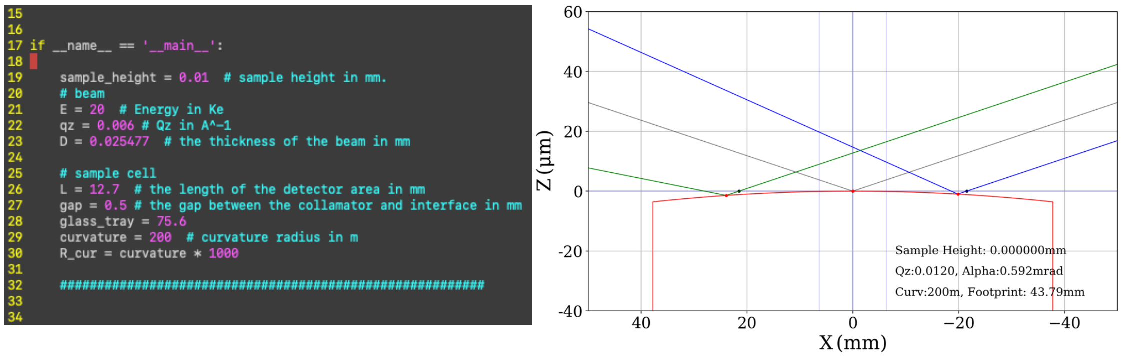
Please check parameters to fit

Figure 21. Dialogs for setting limits on varying parameters (left), and dialogs for setting varying steps for the parameter for which you want to calculate the error bar.

(describe the fitting procedure)

c. Geometry plot

Another feature to add to the software is plotting out scattering geometry of the interface. This feature is currently achieved by running “flu_geometry_2.py”. One can change the geometric parameters in the file. The output is shown in Figure 22.



```
if __name__ == '__main__':

    sample_height = 0.01  # sample height in mm.
    # beam
    E = 20  # Energy in Ke
    qz = 0.006 # Qz in A^-1
    D = 0.025477  # the thickness of the beam in mm

    # sample cell
    L = 12.7  # the length of the detector area in mm
    gap = 0.5 # the gap between the collamator and interface in mm
    glass_tray = 75.6
    curvature = 200  # curvature radius in m
    R_cur = curvature * 1000

    ########################################################

```

Figure 22. Plots for the scattering geometry created by “flu_geometry_2.py”. x axis is the horizontal dimension and y axis is height enlarged by 1000 times. Critical information such as Q_z, incident angle sh, curvature and footprint size are annotated to the lower right corner.

4 NANOSCALE VIEW OF ASSISTED ION TRANSPORT ACROSS THE LIQUID-LIQUID INTERFACE

This chapter was originally a paper published on PNAS under PNAS license (PNAS 2019 116 (37) 18227-18232). According to PNAS copyright policy (Appendix B), author has the right to include this paper into this thesis.

4.1 Introduction

The transfer of metal ions from aqueous to organic phases underlies the process of solvent extraction. Ongoing developments of this process are aimed at optimizing the efficiency and kinetics of the separation and recovery of base, rare earth, and precious metals (1), as well as the reprocessing of spent nuclear fuel and nuclear waste (2). In the latter case, for example, the efficient separation of trivalent minor actinides (Am/Cm) from lanthanides in spent nuclear fuel would reduce the demands imposed on the geological repositories proposed for the long-term storage of nuclear waste (2). Although the interaction of metal ions with solutes at the organic-aqueous interface is likely to determine the efficiency and kinetics of extraction (3), little is known about the mechanism of ion transport across this interface. Conventional hydrodynamic analysis assumes that ions diffuse across the interface on the nanoscale, although interfacial instabilities are predicted and observed on larger spatial scales (4). Here, we extend recent investigations of solvent extraction on the nanoscale to relate the observed interfacial intermediates to the extraction mechanism and efficiency, as well as suggest a role for nonequilibrium interfacial instabilities on the nanoscale.

Metal ion extraction is assisted by the formation of supramolecular complexes with a soluble organic extractant (5). Acidic organo-phosphorus extractants are used extensively and will be studied here (1). They are amphiphilic molecules with a phosphoric acid head group that binds to metal ions and hydrophobic alkyl tail groups that provide sufficient solubility in the organic phase. After extraction into the organic phase, metal ions are found in supramolecular ion-extractant complexes in the form of either coordination complexes or reverse micelles (6-8).

Studies of the kinetics of metal ion extraction indicate that ions and extractants interact at or near the organic-aqueous interface (3, 9). Different authors have suggested that the extractant binds metal ions either in the aqueous phase near the interface, or in the organic phase near the interface, or at the interface itself. For instance, the mass transfer with chemical reaction (MTWCR) mechanism (9) postulates that acidic extractants are transferred into the aqueous boundary layer near the organic-aqueous interface, where they are deprotonated and interact with metal ions to form aqueous ion-extractant complexes, which subsequently diffuse into the organic phase. MTWCR has achieved partial success in describing the extraction kinetics of divalent metal ions with organo-phosphorus extractants (10, 11). On the other hand, it has been suggested that when the interface is occupied by stronger amphiphiles that exclude the extractants, the extractants in the organic phase near the interface can form ion-extractant complexes when fingers of water that contain metal ions reach into the organic phase (12). Somewhat between these two cases, there is evidence that the amphiphilic character of extractants leads to their interaction with ions directly at the interface (13-16); these studies include the suggestion that reverse micelles of extractants that enclose metal ions can form at the interface (17). Largely missing from these investigations has been the application of experimental techniques that can locate and identify metals, extractants, and ion-extractant complexes in the liquid-liquid interfacial region, although recent X-ray and neutron scattering studies have begun to do just that (18-22).

Here, we use interface-sensitive X-ray scattering and fluorescence techniques to locate and identify interfacial species in model extraction systems. The challenge of measuring fast ion transport processes with slow X-ray techniques formerly led us to develop a thermal process that switches between fast and slow rates of extraction (19). This allowed us to characterize intermediate states in the extraction process of a trivalent lanthanide Er(III) and a divalent main group ion Sr(II). One such state consisted of supramolecular erbium-extractant complexes condensed into an inverted bilayer structure in the form of a two-dimensional layer of Er ions sandwiched between two layers of extractant (19). These Er-extractant complexes were formed at the dodecane-water interface within the timescale of minutes used to toggle the thermal switch, and possibly much faster. Similar experiments with strontium ions revealed, instead, a conventional monolayer of extractants with bound Sr(II) ions located on the aqueous side of the interface (20), with no obvious route to transfer ions into the organic phase. Nevertheless, toggling the thermal switch led to extraction of both types of ions, although a larger fraction of erbium than strontium was extracted.

The unexpected structure of inverted bilayers at an organic solvent-water interface is confirmed by the X-ray studies presented here of a model solvent extraction system with Y(III) cations. The inverted bilayer structure contains hydrophobic alkyl tail groups in direct contact with water—illustrated later in Figure 23—instead of the conventional arrangement of amphiphiles in which polar head groups are in contact with water. The stability of this unconventional interfacial structure is confirmed by molecular dynamics (MD) simulations. Comparing the results from these three ions, Y(III), Er(III), and Sr(II), suggests that the electronic configuration of the ion is secondary or irrelevant, whereas the ion charge or oxidation state plays the primary role in determining the extraction mechanism. The form of the interfacial ion-extractant complexes provides insight into the mechanism of ion transport across the interface, which is discussed in the context of dynamical distortions of the interface.

In recent years, MD simulations have clarified the role of interfacial fluctuations in the case of ion transport across bare electrochemically controlled interfaces, although the mechanism of transport under these conditions remains under investigation (23). The degree to which interfacial fluctuations play an important role in the extractant-assisted ion transport processes discussed here is also an open issue.

4.2 Results

Our experimental system consists of a macroscopically flat liquid-liquid interface between a dilute, acidic aqueous solution of metal chlorides [where the metal ion is Y(III), Er(III), or Sr(II)] and a dilute organic solution of the extractant bis(hexadecyl) phosphoric acid (DHDP) ($[CH_3(CH_2)_{15}O]_2POOH$) (10^{-4} M) in n-dodecane [$CH_3(CH_2)_{10}CH_3$]. In the absence of metal ions, the extractant DHDP will form a high-density, ordered monolayer at the dodecane-water interface below an adsorption temperature $T_0 = 38.2$ °C, which varies with pH (19). Under these conditions, DHDP molecules are close-packed with the long axis of the molecule oriented perpendicular to the interface, similar to those shown in Figure 23D. As the temperature is raised above T_0, DHDP desorbs from the interface, leaving behind a disordered partial monolayer (19). When metal ions are present, we have shown previously that changing the temperature from above to below to retards the passage of ions from the water to the dodecane phase, thereby acting as a thermal switch for ion extraction; similarly, the rate of extraction is enhanced if the temperature is raised from below to above T_0 (19). By preparing a model extraction system under conditions with normal rates of extraction ($T > T_0$), then reducing the temperature to below T_0 to retard the rate of extraction, ion-extractant complexes formed in the midst of extraction when $T > T_0$ can be trapped at the interface when $T < T_0$. This metastable interfacial state is then characterized with X-ray reflectivity and X-ray fluorescence near total reflection (XFNTR).

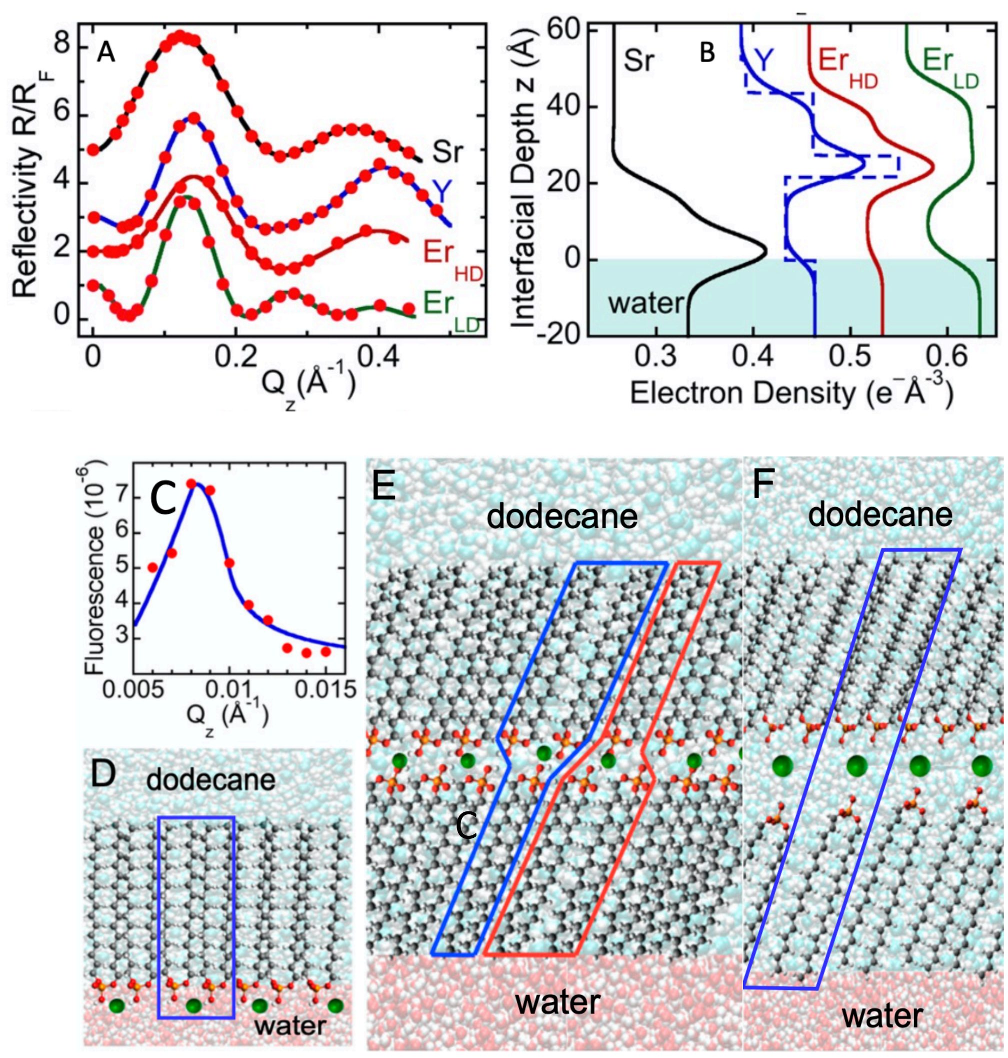


Figure 23 (A) The variation of X-ray reflectivity RðQz Þ with wave vector transfer Qz (perpendicular to the interface) normalized to the calculated Fresnel reflectivity RFðQzÞ, as measured from the interface between metal (Y, Er, Sr) chlorides in water (pH 2.5 for Y and Er, pH 5.3 for Sr) and 10−4 M DHDP in dodecane. Samples were prepared as described in the text and measured at a temperature a few degrees below each sample's adsorption transition To. The upper three curves were shifted for clarity, although R=RF → 1 as Qz → 0 for all measurements. Curves labeled ErHD and ErLD refer to high-density and low-density Er interfaces. Lines are the best fits to the model described in the text. (B) Electron density profiles determined by the fits in A, where the right three curves were shifted for clarity, although ρwater → 0.333e ·Å as z → − 20 Å for all curves before shifting. The profiles are rounded as the result of capillary wave roughness of the interface; the dashed line for Y shows an example of the underlying zero-roughness profile. (C) X-ray fluorescence near total internal reflection (XFNTR) data (dots) and analysis (line) from an interface between 10−4 M DHDP in dodecane and 3 × 10−7 M YCl3 in water (pH 2.5) at 28 °C. Error bars (±1 SD) are generally smaller than or similar to the size of the dots in A and C. (D-F) Molecular representations of the interfacial structures with zero interfacial roughness. (D) Cartoon of the measured monolayer with Sr(II). (E) Cartoon of a hypothetical maximum density inverted bilayer. (F) Cartoon of the measured low-density (LD) inverted bilayer of DHDP with Er(III). High-density (HD) inverted bilayers containing Y(III) or Er(III) consist of an intermediate

configuration of ion-extractant complexes to those shown in D and E, as described in the text. Red and blue boxes identify the ionextractant complexes; red indicates the “up” orientation, and blue is the “down” orientation. Panels E and F are modified and reprinted with permission from ref. 19 (Copyright 2014, American Chemical Society).

Figure 23A shows X-ray reflectivity data, which measures the electron density profile of the liquid-liquid interface (Figure 23B), and cartoons that represent molecular ordering at the interface. Two different structures were measured for multiple Er samples: a structure labeled Er_{HD} that contains a high-density of Er ions and one labeled Er_{LD} with a low density of Er ions. Inverted bilayer structures at the interface (Figure 23E and F) can be identified by a unique experimental signature in the low Q_z region of the data, as exhibited in Figure 23A by the Y(III) and Er(III) samples. As shown previously, X-ray reflectivity data of this form cannot be due to a conventional monolayer, bilayer, or trilayer with DHDP head groups exposed to the water subphase (19). Instead, the head groups are located in the middle of an inverted bilayer (Figure 23 E and F). As shown in Figure 23B for the Y and Er_{HD} structures, the peak in electron density in the middle of the bilayer structure is a region of high electron density, which is due to the location of electron-dense components [phosphoric acid head groups and Y(III) or Er(III) ions]. The uneven shoulders of this peak represent the alkyl chains of the two DHDP layers. The absence of a peak just at the interface, at $z \approx 0$, indicates an absence of DHDP head groups that are directly exposed to the water subphase, in contrast to the conventional expectation that polar head groups of amphiphiles should interact with water. On the other hand, measurements of systems with Sr(II) exhibit a conventional monolayer (Figure 23D) with a peak in the electron density at $z \approx 0$ representing Sr ions bound to DHDP head groups exposed to the water subphase, as well as a shoulder representing tail groups exposed to the dodecane (Figure 23B).

Quantitative analysis of the X-ray reflectivity utilizes a representation of the interfacial structure in terms of slabs of uniform electron density, an example of which is shown in Figure

23B as a dashed-line profile, which is roughened by capillary waves to produce the solid lines in Figure 23B. Each slab represents a different region of the interfacial layer, such as the head groups or tail groups. Table IV shows that most of the tail group electron densities vary from 0.30 to 0.33 $e^{-}\text{Å}^{-3}$, values that are characteristic of all-trans close-packed crystalline or rotator phases of alkanes (24). Exceptions include the terminal portion of DHDP tail groups in the upper leaflet (in contact with dodecane), which has a lower electron density that reveals molecular disorder near the end of the alkyl chains (24). The other exception is the lower density liquid-like tail groups observed in the lower leaflet of the Er_{LD} measurement. Table IV and Figure 23B show that inverted bilayers containing Y ions were structurally similar to the high-density Er_{HD} inverted bilayers.

The tail group thickness of the DHDP monolayer bound to Sr(II) is 20 ± 1 Å, given by the sum of the thicknesses of slabs 2 and 3, which matches the all-trans length, 20.6 Å, of the DHDP tail groups. This indicates that the tail groups are arranged as shown roughly in Figure 23D, although the disorder near the terminal methyl group is not shown. Similar thickness values are observed for the lower leaflet of the Y and Er_{HD} inverted bilayers. Smaller thicknesses of 18-19 Å, as observed for the low-density Er inverted bilayer, may correspond to chains tilted on the order of 25° from the interfacial normal. Even smaller values, as observed for slab 3 of the Y and Er_{HD} inverted bilayers, require a fourth slab to account for the rest of the tail group. In these cases, good fits to the data require four slabs.

Additional information on the ionic content of the interface is provided by XFNTR, which measures the total number of a specific ion (Y, Er, or Sr) per interfacial area (25, 26). Although ions within a distance of roughly 15 nm from the interface can contribute to this measurement, the small bulk concentration of ions ($\sim 10^{-7}$ M for Y and Er solutions) limits the measurable signal to those ions within the interfacial structure. The line shown in Figure 23C is the result of data analysis described in Appendix,_which yields the interfacial area per Y, $A_Y = 68 \pm 3$ Å^2. Previously published values include 81 ± 5 Å^2 for the low-density Er_{LD} inverted bilayer and 90 ± 9 Å^2 for the Sr monolayer (at pH 5.3) (19, 20). By combining XFNTR measurements of the area per ion with X-ray reflectivity results, the analysis described in Appendix, shows that the head group region of the inverted bilayer contains enough electrons to account for three DHDP head groups for each metal ion (Y or Er) plus roughly 0 to three water molecules.

A molecular interpretation of the Y inverted bilayer is the result of modeling that is constrained by the measured values of electron density and thickness of each slab (from X-ray reflectivity), the measured area per Y ion (from X-ray fluorescence), and the constraint that the upper and lower leaflets must occupy equivalent interfacial areas. An earlier analysis, subject to similar constraints, of the low-density inverted bilayer of Er showed that it is equivalent to a condensed state of charge-neutral ion-extractant complexes $Er(DHDP)_3(H2O)_m$ (19), where the structure of the complex is shown in the blue box in Figure 23F. Here, we assume that the Y inverted bilayer comprises charge-neutral ion-extractant complexes $Y(DHDP)_3(H2O)_m$. If all complexes were oriented down (blue boxes in Figure 23 E and F) as in the low-density Er bilayer (Figure 23F), then the area per Y would be twice the area per close-packed DHDP, $A_Y \approx 79$ Å^2(Analysis of Y Inverted Bilayer Structure). If complexes alternate their orientation (with the same number of up and down orientations, as illustrated in Figure 23E), then the area per Y would be 1.5 times the area per DHDP, $A_Y \approx 59$ Å^2. The intermediate value measured by XFNTR for the Y inverted bilayer, $A_Y = 68 \pm 3$ Å^2, suggests an intermediate arrangement. Proceeding beyond this point in the analysis requires an assumption about the mixing of dodecane and DHDP tail groups because our X-ray techniques do not distinguish between them. If we assume that the lower leaflet consists of only DHDP tail groups, but that dodecane can mix into the upper leaflet, then the analysis presented in Analysis of Y Inverted Bilayer Structure, shows that there are 1.5(5) dodecane molecules for each DHDP molecule in slabs 3 and 4 that model the upper leaflet and the fraction of complexes pointing down is 0.3(1). Although other assumptions about the location of dodecane within the inverted bilayer lead to different numerical values, the analysis shows that the structure of the Y inverted bilayer can be described as a condensed phase of ion-extractant complexes at the interface. Although the lack of XFNTR data for the observation of the high-density Er inverted bilayer does not allow for this type of analysis, the similarity of its electron density profile to the Y system suggests that it too can be described as a condensed phase of interfacial ion-extractant complexes.

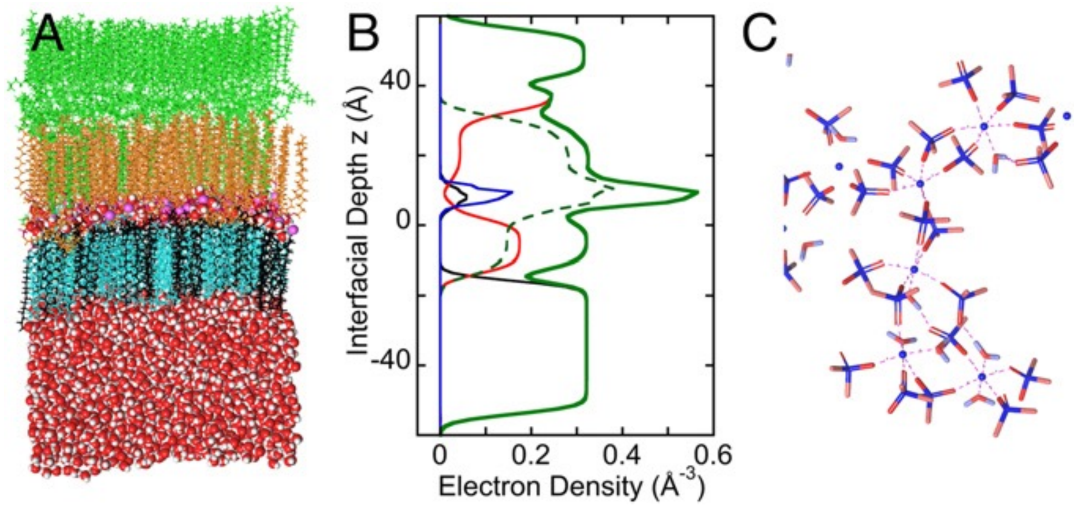


Figure 24. MD simulation results. (A) Snapshot of inverted bilayer from the last frame of the simulation—water (Bottom, red and white), dodecane (Top, green), inverted bilayer: ions in blue, dodecane that started in the top leaflet is shown in green, dodecane that started in the bottom leaflet is colored cyan, DHDP molecules that started in the top leaflet are colored gold, and DHDP that started in the bottom leaflet is colored black. Topmost layer of dodecane is ordered at the vapor interface, which is not relevant for comparison with the results of X-ray measurements. A smaller, disordered layer of dodecane exists immediately adjacent to the inverted bilayer. (B) MD electron density profile averaged over the final 100 ns of the simulation: total density profile in solid green, water in black, dodecane in red, Er ions in blue, and DHDP in dashed green. (C) Er coordination showing only DHDP head groups and water molecules.

We performed classical MD simulations to investigate the stability of the inverted bilayer structure. Figure 24A shows the last frame of this simulation, and Figure 24B shows the time-averaged electron density profile. Both illustrate the qualitative features of the measured electron density profiles of the high-density inverted bilayers containing Y and Er (Figure 24B), namely, a high-density peak from the head groups and metal ions, with shoulders from the DHDP tail groups in the upper and lower leaflets of the bilayer. The experimental electron density profiles appear more disordered because they are roughened by capillary wave fluctuations of the interface (σ in Table IV), which have a much smaller effect on profiles calculated from small simulation boxes. This roughening will smear the dips in electron density at the bottom and top of the bilayer observed in the simulations, but not observed in the experiments.

Table IV. Best-fit parameters to the X-ray reflectivity data

Ion	σ_1(Å)	d_1(Å)	$\rho_1(e^{-}\text{Å}^{-3})$	d_2(Å)	$\rho_2(e^{-}\text{Å}^{-3})$	d_3(Å)	$\rho_3(e^{-}\text{Å}^{-3})$	d_4(Å)	$\rho_4(e^{-}\text{Å}^{-3})$
Y	3.1(2)	21.5(2)	0.303(1)	6(1)	0.42(1)	16(1)	0.332(1)	9(1)	0.265(2)
Er_{HD}	3.4(3)	20.6(1)	0.318(1)	8(2)	0.40(2)	15(2)	0.319(5)	5(2)	0.25(1
Er_{LD}	3.6(3)	18.9(7)	0.279(3)	9(3)	0.33(1)	18(3)	0.324(3)	—	—
Sr	4.3(2)	4(2)	0.6(1)	17(1)	0.333(2)	3(1)	0.21(3)	—	—

Fits to data from interfaces between 10^{-4} M DHDP in dodecane and (line 1) 3 × 10^{-7} M YCl3 in water (pH 2.5) at 28 °C; (line 2) 10^{-7} M ErCl3 in water (pH 2.5) at 28 °C with a high density of Er ions; (line 3) 5 × 10^{-7} M ErBr3 in water (pH 2.5, adjusted with HBr) at 28 °C (19) with a low density of Er ions; (line 4) 10−5 M SrCl2 in water (pH 5.3, adjusted with acetate buffer) at 36.7 °C (20). The electron densities of the bulk aqueous and organic phases are $0.333e^{-}\text{Å}^{-3}$ (28 °C) and 0.2574 $e^{-}\text{Å}^{-3}$, respectively. The thicknesses of the four slabs (d_1, d_2, d_3, and d_4), the electron densities of the slabs (ρ_1, ρ_2, ρ_3, and ρ_4), and the interfacial roughness σ are fitting parameters. For the Y and Er samples: slab 1 represents the DHDP tail groups of the lower leaflet (in contact with water) of the inverted bilayer, slab 2 represents the DHDP head group region that includes metal ions and possibly water, and slabs 3 and 4 represent the tail groups of the upper leaflet (in contact with dodecane). For the Sr sample: slab 1 represents the DHDP head group and Sr ions, and slabs 2 and 3 represent the DHDP tail group. Parenthetical numbers represent 1 SD in the last significant digit.

These simulations were designed to test the stability, but not the formation, of the low-density Er inverted bilayer. The initial state of the simulation placed one-half the number of DHDP molecules in the lower leaflet than in the upper leaflet; the extra space in the lower leaflet was filled with dodecane. This led to a higher electron density in the lower leaflet than observed in experiments of the low-density Er inverted bilayer, but similar to the electron density measured in the high-density inverted bilayer. The simulations demonstrated that the structure of the inverted bilayer is stable at the interface for the 100-ns span of the simulation, with occasional motion of dodecane molecules between the two leaflets.

Figure 24C shows a mesh-like coordination of Er with water molecules and DHDP head groups. The low density of Er ions used for the simulation (86 Å^2 per Er) produces blank regions in Figure 24C. The geometry of Er-O coordination is octahedral (Figs. S2 and S3) with coordination to six oxygen atoms, as observed in Er-DHDP complexes that have been extracted into the bulk dodecane (19). Radial distribution functions from the Er ion (Figs. S4 and S5) demonstrate that bond lengths are different in the inverted bilayer and in bulk extracted Er-DHDP complexes (19), most likely as the result of constraints placed upon the geometrical arrangement of the DHDP tail groups in the inverted bilayer.

4.3 Discussion

4.3.1 Static Interfacial Structures.

The measurements reported here characterized the static structure of interfacial states formed in the midst of solvent extraction, as ions are transported across the water-dodecane interface. They show that small rare earth ions with oxidation state +3, Y(III) and Er(III), form inverted bilayers at the liquid-liquid interface upon thermally retarding the extraction process. Although we have observed different variations of the inverted bilayer, all have a single layer of ions sandwiched between back-to-back layers of DHDP extractants. The X-ray measurements are consistent with a model of the inverted bilayer as an interfacial condensed state of supramolecular ion-extractant complexes of the form $M(DHDP)_3(H2O)_m$, where M is Er or Y and m varies roughly from 0 to 3. Although the inverted bilayer is not expected to be an equilibrium state, since it was formed under nonequilibrium conditions, MD simulations confirmed the short-term stability of this structure.

Studies of Sr(II) extraction with DHDP under similar conditions exhibited a different intermediate state, in which Sr(II) ions remained in the water phase and were bound (or located adjacent) to the head groups of an interfacial monolayer of DHDP. Comparison of the intermediate states of these three ions, Y(III), Er(III), and Sr(II), suggests that the oxidation state of the ion, or possibly just ionic charge, is the primary factor that determines the form of the intermediate state. These studies also allow for a comparison of the effect of electronic configuration on the intermediate state. We note that Sr(II) and Y(III) have the same closed-shell electronic configuration ($4p^6$) of the unreactive noble gas Kr, whereas Er(III) has a more complex $4f^{11}$ electronic configuration. It appears that the electronic configurations of these ions are not the determining factor in the structure of the intermediate state, in contrast to previous suggestions from kinetic studies of divalent ion extraction (17). Further studies are required to explore other effects. For example, since the ionic radius of Sr(II) (118 pm) is substantially larger than that of either Er(III) (89 pm) or Y(III) (90 pm), which are roughly equal, future studies will explore the role of ion size (27).

4.3.2 <u>Consequences for the Mechanism of Extraction.</u>

Our results demonstrate that Y(III) and Er(III) are more effective at coordinating with DHDP than Sr(II). For instance, the formation of inverted bilayers containing Y(III) and Er(III), as well as their extraction at temperatures above the adsorption transition To, were observed with pH 2.5 water for which 94% of the phosphoric acid head groups would have been protonated and uncharged in the absence of metal ions (Fraction of Protonated Head Groups in DHDP

Monolayer). However, Sr(II) binding to the charge-neutral DHDP monolayer under similar low-pH conditions was not observed (20). Instead, Sr(II) binding was observed only at higher pH, with one Sr(II) for every two DHDP measured at pH 5.3 and higher pH values. Even under these conditions of Sr saturation of the interface, combined X-ray reflectivity and XFNTR results show that approximately one-third of the interfacial Sr(II) ions are not closely bound to the DHDP head groups, but exist only in a diffuse electrical double layer near the interface (20).

Y(III) and Er(III) are also more efficiently extracted from the aqueous phase than Sr(II). Analysis of the metal content in the aqueous phase before and after extraction by inductively coupled plasma atomic emission spectroscopy (ICP-AES) and ICP mass spectroscopy showed that 87 (3)% of the Y(III) was extracted at 50 °C, more than 80% of the Er(III) was extracted at 55 °C (19), but only 45% of Sr(II) was extracted at 50 °C (pH 5.3) (20).

The interfacial state of Y(III) and Er(III) sandwiched between layers of DHDP extractants suggests the prompt transfer of these cations from the aqueous side of the liquid-liquid interface to a coordinated ion-extractant environment on the organic side. These ion-extractant complexes represent an intermediate state in which ions have been transported across the aqueous-organic interface, but have not yet been dispersed in the organic phase. In contrast to this, the observation of a conventional monolayer of DHDP extractants with Sr(II) bound to DHDP head groups, but remaining in contact with the water phase, suggests a slower kinetics of transfer of Sr(II) from water to dodecane, whose mechanism involves at least one additional step to transport the ion across the aqueous-organic interface.

Insight into this additional step may be provided by small angle neutron scattering measurements by Steytler et al. (6) of the metal salts $M^{n+}(DEHP)_n$ dissolved in cyclohexane. The extractant bis(2-ethylhexyl) phosphoric acid (DEHP) has shorter, branched chains, but the same head group as DHDP. Steytler et al. studied the trivalent ion Al(III) and several divalent ions, including Ca(II) that is similar to the Sr(II) studied here. Although they observed small spherical complexes of Al-DEHP that were similar in size to Er-DHDP complexes that had been fully extracted into bulk dodecane (19), they observed larger rod-like reverse micelles of several different divalent ions, including Ca(II).

If reverse micelles form in our Sr(II)-DHDP extraction system, then budding of the micelle at the interface could be the additional, unobserved step in the ion transport across the aqueous-organic interface (17). A plausible mechanism for this process consists of three stages: (i) DHDP adsorption onto the interface and binding to Sr(II) ions, (ii) formation of interfacial domains of $Sr(DHDP)_2$ complexes, and (iii) domain budding of reverse micelles into the organic phase. To explore the plausibility of budding of reverse micelles at the interface, we consider circular domain budding into a spherical reverse micelle (Figure 25), as described in the following equation introduced by Lipowsky to model the energy E of bud formation in biomembranes (28):

$$E = E_{bend} + E_{edge} = 2\pi\kappa\left(LC - LC_{sp}\right)^2 + 2\pi\lambda L\sqrt{1-(LC/2)^2} \tag{4.1}$$

The first term in Equation (4.1) is the energy Ebend required to bend the domain into a spherical cap, or full sphere, of curvature C different from its spontaneous curvature C_{sp}. The domain is further characterized by its bending rigidity κ and domain area πL^2 (Figure 25). Spontaneous curvature of the domain can arise from the asymmetry of the monolayer-containing interface, which has ions on one side interacting with extractants on the other. The second term in Equation (4.1) expresses the energy E_{edge} of the domain edge in terms of its line tension λ, where dashed lines in Figure 25 illustrate the domain edge. Formation of a complete spherical bud leads to the extraction of enclosed ions when the bud separates from the interface and goes into the organic phase (Figure 25D).

A minimum size L^o is required for the domain to form a complete bud. A domain can increase its size to this value by the aggregation of interfacial Sr(DHDP)$_2$ complexes. The larger domain size that results from this aggregation has a longer domain edge and, consequently, larger edge energy. This larger edge energy can be recovered: as the domain bends to form a more complete sphere, illustrated by progressing from Figure 25B to Figure 25C, the reduction in edge length reduces the edge energy. This reduction balances the cost in bending energy required to form a spherical bud.

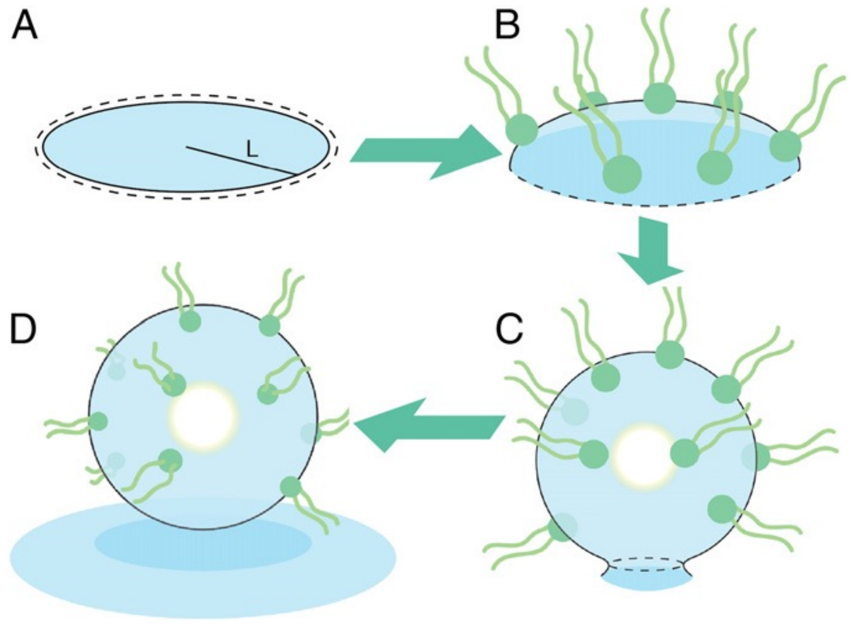


Figure 25. Domain budding mechanism. (A) A flat region of bare interface (dodecane above, water below) becomes (B) spontaneously curved due to the adsorption of extractants and their interactions with ions (not shown) at the interface. (C) The reduction in length of the domain edge (dashed line) reduces the line tension energy, which balances the bending energy required to form a spherical reverse micelle. (D) Separation of the micelle from the interface extracts the ions (not shown) in the interior of the reverse micelle into the bulk organic phase.

Lipowsky showed that complete budding is energetically favorable for domain sizes $L \geq L^o$, where $L^o = 8\kappa/\lambda\left[1 + \left(4\kappa|C_{\mathrm{sp}}|/\lambda\right)^{2/3}\right]^{3/2}$ (28). Literature values for κ, λ, and C_{sp} for compounds similar to our extractant DHDP are discussed in Appendix, and yield a range of values for the ratio κ=λ, given by $4\mathrm{nm} \leq \kappa/\lambda \leq 20\mathrm{nm}$, and for C_{sp}, given by $0.1\mathrm{nm}^{-1} \leq C_{\mathrm{sp}} \leq$

0.3nm^{-1}. These values produce a range of limiting lengths, $4\text{nm} \leq L^o \leq 14\text{nm}$, whose lower value of 4 nm describes a bud that contains the same number of extractants as the aggregation number of the reverse micelles measured by Steytler et al. (6). Note that the DEHP studied by Steytler et al. will have values of C_{sp} at the higher end of the stated range, thereby leading to values of L^o at the lower end of our prediction. These calculations suggest that Sr-DHDP reverse micelles can be formed at the interface by spontaneous budding, although additional experiments are required to confirm this result.

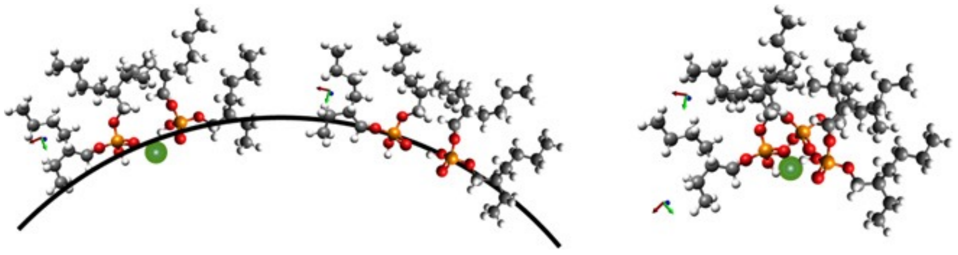


Figure 26. Cartoon of the interaction of a bulky branched-chain extractant DEHP with (A) a divalent ion (and two DEHP molecules) and (B) a trivalent ion (and three DEHP molecules) at the liquid-liquid interface (represented by the line), which illustrates how the interaction with the ion produces a spontaneous curvature of the interface.

Equation (4.1) indicates that interfacial ion-extractant complexes that produce larger values of spontaneous curvature C_{sp} will require less bending energy to make a complete bud. Larger values of C_{sp} may result from the interaction of bulky extractants with ions at the interface—these include extractants that are commonly used in solvent extraction processes, like DEHP that has branched alkyl tail groups (Figure 26A), or malonamides and diglycolamides (29) that have bulky head groups. Bulky extractants form smaller complete buds since the spontaneous curvature is closer to the value of curvature $C = 2/L$ required to make a complete spherical bud. Higher oxidation state ions that coordinate a larger number of extractants are expected to produce an even larger spontaneous curvature (Figure 26B). This physical picture suggests the formation of small supramolecular complexes relevant to the extraction of Y(III) and Er(III). Further research is required to establish a quantitative relationship between the shape and chemical properties of the extractant molecule, the interfacial elastic properties—κ, λ, and C_{sp}—and the extraction kinetics.

4.4 Materials and Methods

4.4.1 Materials and Sample Cell.

N-Dodecane [$CH_3(CH_2)_{10}CH_3$] (>99%; Sigma-Aldrich) and bis(hexadecyl) phosphoric acid (DHDP) [$CH_3(CH_2)_{15}O]_2POOH$] (>98%; SigmaAldrich) (Scheme 1) were purified as described previously (19). Aqueous solutions of yttrium chloride hexahydrate ($YCl_3{\cdot}6H_2O$) (>99.99%; Sigma-Aldrich) were purified as described previously for $ErBr_3$ (19). Water was produced by a Nanopure UV Barnstead system. Hydrochloric acid (Optima grade; Fisher Scientific) was used to adjust the pH values of YCl_3 solutions. Dodecane-water interfaces (2.2:1 volume ratio) were temperature controlled (±0.03 °C) (20).

4.4.2 Fraction Extracted.

Aqueous and organic phases were heated to 50 or 55 °C, placed into contact in a glass dish, and sat for periods varying from 1/2 to 24 h. A portion of the aqueous phase was extracted and analyzed by ICP-MS (Galbraith Laboratories) for Y or by ICP-AES for Er and Sr.

4.4.3 X-Ray Reflectivity and XFNTR.

X-ray measurements from liquid-liquid interfaces were made at ChemMatCARS Sector 15 of the Advanced Photon Source at an X-ray energy of 20 keV. X-ray reflectivity $R(Q_z)$ was measured as a function of wave vector transfer normal to the interface $Q_z = (4\pi/\lambda_x)\sin\alpha$, where λ_x is the X-ray wavelength and α is the angle of incidence. The reflected intensity (with background subtracted) was normalized to the incident intensity. The $R(Q_z)/R_F(Q_z)$ data in Figure 23A represent the measured reflectivity normalized to the calculated Fresnel reflectivity (30). Data are analyzed with a slab model described previously (19, 30). XFNTR data consisted of measurements of fluorescence spectra for values of Q_z slightly below and above the condition for total reflection (Fig. 1C). Measurement and analysis methods were published previously (20).

4.4.4 MD Simulations.

Classical MD simulations were performed with Schrodinger Desmond (academic release) using the OPLS-2015 force field (31). The inverted bilayer system was built from two back-to-back monolayers on a 65.6 × 65.6-Å^2 rectangular grid (x-y) of 100 DHDP (initial all-trans state) with phosphate head groups in the x-y plane and tail groups oriented along the z axis. Fifty DHDP molecules were removed from the lower leaflet, and dodecane added into the intervening spaces. Monolayer leaflets were separated by 8 Å, which was filled with 50 Er(III) ions and 150 water molecules to mimic the low-density Er inverted bilayer. A preequilibrated slab of SPC-E water molecules was placed in proximity to the tail groups of the lower leaflet and a preequilibrated slab of dodecane molecules was placed in proximity to the tail groups of the upper leaflet. The box dimension in the z direction was set to 200 Å, allowing for two liquid-vapor interfaces at the top and bottom of the periodic box. Equilibration occurred in stages, first by restraining the ions and head groups of the surfactant with a harmonic potential and allowing the water and dodecane to equilibrate. Harmonic restraints were then removed and further equilibration was performed for 50 ns before data production for 100 ns commenced.

4.4.5 Acknowledgments

We acknowledge support from US Department of Energy (DOE), Office of Basic Energy Sciences (OBES) (DE-SC0018200) (to M.L.S. and I.B.), the National Science Foundation (CHE-1363076) (to I.B.), the University of Illinois at Chicago (UIC) Department of Physics (M.L.S.), UIC Contest (J.S.H.), UIC Chancellor's Undergraduate Research Award (to D.J.W.), and support of earlier studies (19, 20) of Er and Sr through US DOE, OBES, Chemical Sciences, Geosciences, and Biosciences Division, under Contract DE-AC02-06CH11357 (subcontract to M.L.S. from Dr. Lynda Soderholm). This research used resources of the Advanced Photon Source, a US DOE Office of Science User Facility operated for the DOE Office of Science by Argonne National Laboratory under Contract DE-AC02-06CH11357. ChemMatCARS is supported by National Science Foundation Grant CHE-1346572.

4.5 Cited Literature

1. Tasker PA, Plieger PG, West LC (2004) Metal complexes for hydrometallurgy and extraction. Comprehensive Coordination Chemistry II: From Biology to Nanotechnology, eds McCleverty JA, Meyer TJ (Elsevier, Oxford), Vol 9, pp 759-808.
2. Lumetta GJ, Nash KL, Clark SB, Friese JI (2006) Separations for the Nuclear Fuel Cycle in the 21st Century, ACS Symposium Series 933 (American Chemical Society, Washington, DC).
3. Danesi PR, Chiarizia R (1980) The kinetics of metal solvent extraction. Crit Rev Anal Chem 10:1-126.
4. Sternling CV, Scriven LE (1959) Interfacial turbulence: Hydrodynamic instability and the Marangoni effect. AIChE J 5:514-523.
5. Wilson AM, et al. (2014) Solvent extraction: The coordination chemistry behind extractive metallurgy. Chem Soc Rev 43:123-134.
6. Steytler DC, Jenta TR, Robinson BH, Eastoe J, Heenan RK (1996) Structure of reversed micelles formed by metal salts of bis(ethylhexyl) phosphoric acid. Langmuir 12:1483-1489.

7. Gannaz B, Antonio MR, Chiarizia R, Hill C, Cote G (2006) Structural study of trivalent lanthanide and actinide complexes formed upon solvent extraction. Dalton Trans 38: 4553-4562.

8. Ellis RJ, Anderson TL, Antonio MR, Braatz A, Nilsson M (2013) A SAXS study of aggregation in the synergistic TBP-HDBP solvent extraction system. J Phys Chem B 117: 5916-5924.

9. Hughes MA, Rod V (1984) A general model to account for the liquid/liquid kinetics of extraction of metals by organic acids. Faraday Discuss Chem Soc 77:75-84.

10. Dreisinger DB, Cooper WC (1989) The kinetics of zinc, cobalt, and nickel extraction in the D2EHPA-heptane-HClO4 system using the rotating diffusion cell technique. Solvent Extr Ion Exch 7:335-360.

11. Dreisinger DB, Cooper WC (1986) The kinetics of cobalt and nickel extraction using HEHEHP. Solvent Extr Ion Exch 4:317-344.

12. Qiao B, Muntean JV, Olvera de la Cruz M, Ellis RJ (2017) Ion transport mechanisms in liquid−liquid interface. Langmuir 33:6135-6142.

13. Hunt EC (1969) The interaction of alkyl phosphate monolayers with metal ions. J Coll Int Sci 29:105-115.

14. Szymanowski J (2000) Kinetics and interfacial phenomena. Solvent Extr Ion Exch 18: 729-751.

15. Testard F, Berthon L, Zemb T (2007) Liquid-liquid extraction: An adsorption isotherm at divided interface? C R Chim 10:1034-1041.

16. Watarai H (2014) Interfacial molecular aggregation in solvent extraction systems. Ion Extraction and Solvent Extraction: A Series of Advances (CRC, Boca Raton, FL), Vol 21, pp 159-195.

17. Plucinski P, Nitsch W (1992) Kinetics of the interfacial ion exchange in Winsor II microemulsion systems. J. Coll. Int. Sci. 154:104-112.

18. Bu W, et al. (2011) X-ray fluorescence from a model liquid/liquid solvent extraction system. J Appl Phys 110:102214.

19. Bu W, et al. (2014) Observation of a rare earth ion-extractant complex arrested at the oil-water interface during solvent extraction. J Phys Chem B 118:10662-10674.

20. Bu W, et al. (2014) X-ray studies of interfacial strontium-extractant complexes in a model solvent extraction system. J Phys Chem B 118:12486-12500.

21. Scoppola E, et al. (2015) Structure of a liquid/liquid interface during solvent extraction combining X-ray and neutron reflectivity measurements. Phys Chem Chem Phys 17: 15093-15097.

22. Scoppola E, et al. (2016) Solvent extraction: Structure of the liquid-liquid interface containing a diamide ligand. Angew Chem Int Ed Engl 55:9326-9330.

23. Benjamin I (2015) Reaction dynamics at liquid interfaces. Annu Rev Phys Chem 66: 165-188.

24. Small DM (1986) The Physical Chemistry of Lipids (Plenum, New York).

25. Bloch JM, Yun W (1990) Condensation of monovalent and divalent metal ions on a Langmuir monolayer. Phys Rev a 41:844-862.

26. Bu W, Vaknin D (2009) X-ray fluorescence spectroscopy from ions at charged vapor/water interfaces. J Appl Phys 105:084911.

27. Shannon RD (1976) Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides. Acta Crystallogr A 32:751-767.

28. Lipowsky R (1992) Budding of membranes induced by intramembrane domains. J Phys II 2:1825-1840.

29. Ansari SA, Pathak P, Mohapatra PK, Manchanda VK (2012) Chemistry of diglycolamides: Promising extractants for actinide partitioning. Chem Rev 112:1751-1772.

30. Pershan PS, Schlossman ML (2012) Liquid Surfaces and Interfaces: Synchrotron X-ray Methods (Cambridge Univ Press, Cambridge, UK).

31. Bowers KJ, et al. (2006) Scalable algorithms for molecular dynamics simulations on commodity clusters. Proceedings of the ACM/IEEE Conference on Super-Computing (SC06) (ACM, New York), 84.

4.6 Supporting Information

4.6.1 X-Ray Fluorescence near Total Reflection Analysis of Y Samples

X-ray fluorescence near total reflection (XFNTR) data were measured immediately after the reflectivity measurements. The measured variation of fluorescence intensity with Q_z shown in Fig. 1C is determined by the distribution of yttrium. The interfacial density of yttrium is given by I=CTL, where I is the fluorescence intensity from the interface (upon accounting for background fluorescence and fluorescence from the bulk solution as described in ref. 1), C is a scale factor, L is an integration over X-ray paths through the sample, and T is the Fresnel transmission coefficient. To extract the scale factor C described in ref. 1, which takes into account the geometry of scattering, XFNTR data were also measured from a reference sample interface between pure dodecane and a 0.1 M YCl3 aqueous solution. Since the coefficient T has a sharp peak at the critical wave vector transfer Qc, the rounded top in the experimental data shown in Fig. 1C implies that interfacial curvature cannot be ignored. The fit shown by the solid line in Fig. 1C models the interface as a spherical surface with radius of curvature of 100(+35/−65) m. The analysis of XFNTR data shown in 2 Fig.1CyieldsaninterfacialareaperYionAY =68±3Å.

Figure 27 shows an example of the measured fluorescence spectrum from an inverted bilayer containing yttrium at the reflection condition for Qz = 0.014Å^{-1} The peak position of the $K\alpha$ emission line for yttrium located at ~14.9 keV contains both the Kα1 emission at 14.9584 keV and Kα2 at 14.8829 keV (2). Figure 27A shows that the peak is well separated from the direct beam at 20 keV and its Compton peak. The background at the location of the Y Kα line is fit with an exponential (Figure 27B). After background subtraction, the peak is fit to a Gaussian function (Figure 27C). The total fit is shown in Figure 27D. The total intensity is the area under the peak represented by the product of its amplitude and width. The values of intensity as a function of Qz shown in Figure 27C in the main paper were then analyzed as described in ref. 1.

4.6.2 Composition of the Head Group Region in the Inverted Bilayer

The middle region (slab 2) of the inverted bilayer has a thickness d_2=6(1)Å and an electron density ρ_2=0.42(1)Å. Under the assumption that the second slab consists of head groups of DHDP, Y^{3+} or Er^{3+} ions, and water molecules, and that each metal ion is complexed with three DHDP molecules, the total number of electrons per area of the metal ion in slab2 is given by the following:

$$d_2\rho_2A_M = 3e_{\mathrm{PO}_4^-} + e_{M^{3+}} + N_{H_2O}e_{H_2O} \tag{4.2}$$

where eH2O=10, e 3+ =36, eEr3+ =65, and ePO− = 48 are the electron number of each species, and NH2 O is the number of water molecules per area occupied by a metal ion. Using the values in Table1ofthemainpaper,aswellasA =68±3Å2 andA = 2 Y Er 81± 5 Å , indicates that NH2O = 0 ± 2 for the Y inverted bilayer and NH2O = 3 (+10/−3) for the low-density Er inverted bilayer. This calculation cannot be done for the high-density Er bilayer because XFNTR data were not measured for those samples.

4.6.3 Analysis of Y Inverted Bilayer Structure

As described in the main paper, the area per Y ion, A_Y, measured by XFNTR suggests that the Y inverted bilayer has a structure that is intermediate between the maximum density inverted bilayer (Fig. 1E) and the low-density inverted bilayer (Fig. 1F). Since our X-ray measurements cannot distinguish between dodecane and the alkyl tails in DHDP, further analysis to reveal the structure of this highdensity inverted bilayer requires some assumptions about the location of dodecane within the inverted bilayer. Although these assumptions are not unique, the following model is offered as an example of a self-consistent analysis of these data. The results of this model indicate that the measured structure of the Y inverted bilayer can be explained as a condensed phase of ion–extractant complexes of the form $Y(DHDP)_3(H_2O)_m$ with a ratio of down to up complexes intermediate between the 1:1 value of the maximum density bilayer (Fig. 1E) and the 1:0 value of the low-density bilayer (Fig. 1F). Note that Composition of the Head Group Region in the Inverted Bilayer demonstrated that the number of water molecules in the Y ion–extractant complex is $m = 0 \pm 2$.

The model is constrained by the following values:

i. Measured values of electron densities and thickness of each slab (from X-ray reflectivity) as given in Table 1 in the main paper;

ii. Measured area per Y ion (from X-ray fluorescence, XFNTR), $A_Y = 68 \pm 3 Å^2$;

iii. The areas occupied by the upper and lower leaflets have to be the same.

Assumptions are as follows:

i. The lower leaflet (slab 1) consists of only DHDP tails.

ii. The midregion (slab 2) consists of DHDP head groups, Y ions, and possibly water.

iii. The upper leaflet (slabs 3 and 4) consists of DHDP plus dodecane.

iv. AllDHDPandYionsarein3:1complexes(3DHDPto1Y ion) that are oriented either up (2 DHDP in lower leaflet and 1 in upper leaflet; red box in Fig. 1D), or down (2 DHDP in upper leaflet and 1 in lower leaflet; blue box in Fig. 1D). This assumption is suggested by the measurements of the lowdensity Er inverted bilayer, where it was demonstrated that the structure of the inverted bilayer could be explained as a condensed phase of ion–extractant complexes of the form $Er(DHDP)_3(H_2O)_3$. Note that the analysis of the low-density Er inverted bilayer did not require assuming a 3:1 ratio of DHDP to Er, but allowed this to be determined from the data (3).

First, we analyze slab 1. The measured thickness of slab 1, d_1 = 21.5 Å, is within 1 Å of the all-trans length of hexadecyl chains, $L_{\mathrm{trans}} 15 \times 1.27\text{Å}(\mathrm{C-C}) + 1.5\text{Å}(\mathrm{C-H}) = 20.6\text{Å}$. A measure of the area per DHDP using slab 1 is as follows:

$$A_{D1} = 258/(d_1\rho_1) = 39.6\ \text{Å}^2 \tag{4.3}$$

where there are 258 electrons in the two hexadecyl tails of DHDP. This value of A_{D1} is twice the known cross-sectional area of a single all-trans alkyl chain (A_0 = 19.83 Å^2) (4). The measured electron density of slab 1, $\rho_1 = 0.303\ e^{-}\text{Å}^{-3}$, is consistent with electron densities measured for close-packed alkane rotator phases (5).

If all ion–extractant complexes were oriented up, then the area 2 per Y ion would have been $A_{\mathrm{Y}} = 2A_{\mathrm{D1}} \approx 79\ \text{Å}^2$. If complexes alternate their orientation, resulting in the same number of up and 2 down orientations, then $A_{\mathrm{Y}} = 1.5A_{\mathrm{D1}} \approx 59\ \text{Å}^2$. Our XFNTR measurements of the area per Y ion yielded $A_{\mathrm{Y}} = 68 \pm 3\ \text{Å}^2$, suggesting an intermediate arrangement.

Now, we apply the constraints and assumptions listed above to determine this intermediate arrangement. For each up configuration of an ion-extractant complex, let there be x down configurations. Define the "effective area" as the area occupied by an up complex and x associated down complexes. Equating the areas in the lower leaflet and the midregion,

$$(2 + x)A_{D1} = (1 + x)A_{\mathrm{Y}} \tag{4.4}$$

Let there be f dodecane molecules for each DHDP molecule in the upper leaflet, then the area per DHDP/dodecane unit is given by the following:

$$A_U = \frac{258 + 98f}{d_3\rho_3 + d_4\rho_4} \tag{4.5}$$

where there are 258 electrons in the two hexadecyl tails of DHDP and 98 electrons in dodecane.

Equating the effective areas in the lower and upper leaflet yields the following:

$$(2+x)A_{D1} = (1+2x)A_U \tag{4.6}$$

Substituting AU from Eq. S4 into Eq. S5 produces the following:

$$f = \frac{b-258}{98} \tag{4.7}$$

where

$$b = [A_{D1}(d_3\rho_3 + d_4\rho_4)][(2+x)/(1+2x)] \tag{4.8}$$

Now, we solve Eq. S3 for x and Eq. S6 for f to produce the configuration in the inverted bilayer. Solving Eq. S3 for x, the number of down complexes associated with each up complex,

$$\begin{aligned} x &= (A_Y - 2A_{D1})/(A_{D1} - A_Y) \\ &= (68 - 2 \times 39.6)/(39.6 - 68) \\ &= 0.39 \end{aligned} \tag{4.9}$$

We use this value of x to solve for the constant b in Eq. S7,

$$\begin{aligned} b &= [A_{D1}(d_3\rho_3 + d_4\rho_4)][(2+x)/(1+2x)] \\ &= [39.6 \times (16 \times 0.332) + 9 \times 0.265][(2+0.39)/(1+2 \times 0.39)] \\ &= 409 \end{aligned} \tag{4.10}$$

The constant b is substituted into Eq. S6 to solve for the number f = 1.54 of dodecane molecules for each DHDP molecule in the upper leaflet.

This model shows that there are 1.5 ± 0.5 dodecane molecules for each DHDP molecule in slabs 3 and 4 that model the upper leaflet and the fraction of complexes pointing down is given by $x/(1+x) = 0.3 \pm 0.1$. These specific numerical values depend upon the assumption that the lower leaflet consists of only DHDP molecules and the upper leaflet is a mixture of DHDP and dodecane molecules. Other assumptions of this nature are possible, although our data are not consistent with the assumption that the upper leaflet consists of only DHDP. The utility of this model is to show that the measured electron density and ion interfacial density can be explained as a condensed layer of ion–extractant complexes.

4.6.4 Simulation Results

Figure 28 and Figure 29 demonstrate the octahedral coordination of oxygen about erbium in the inverted bilayer. Figure 30 and Figure 31 illustrate the radial distribution functions.

4.6.5 Fraction of Protonated Head Groups in DHDP Monolayer

We consider a monolayer of DHDP at the dodecane−water interface with pH 2.5 water adjusted by adding HCl. Although similar calculations have been presented before, we present it here for the convenience of the reader. According to the Poisson− Boltzmann theory for monovalent ions, the electrical potential is given by the following:

$$\psi(z) = -\frac{2k_BT}{e}\ln\left[\frac{1+\gamma e^{-z/\lambda_D}}{1-\gamma e^{-z/\lambda_D}}\right] \tag{4.11}$$

With Debye screening length $\lambda_D = (\epsilon_0\epsilon_r k_B T/2e^2 n_b)^{1/2} \approx 55$ Å for $n_b = 10^{-2.5} = 3.16 \times 10^{-3}$ Mn, and $\gamma = -\tanh(e\psi(0)/4k_BT)$ determined by the interfacial potential. The boundary condition can be obtained by considering the electric field strength at the interface ($z \to 0^-$) from the potential given above and Gauss's law, expressed as follows:

$$-\frac{\mathrm{d}\psi(z)}{\mathrm{d}z}\Big|_{z\to 0^-} = -\frac{2k_BT}{e\lambda_D}\sinh\frac{e\psi(0)}{2k_BT} = \frac{|\sigma_s|}{\epsilon_0\epsilon_r} \tag{4.12}$$

where σ_s is the interfacial charge density. For a fully deprotonated close-packed DHDP monolayer, $\sigma_s = -e/A$, where A is the molecular area.

However, the proton H+ can bind to and neutralize the charged head group, and as a result, the interfacial charge density is reduced by a factor α. That reaction is given by $\mathrm{H^+ + PO_4^- \Leftrightarrow PO_4H}$, and the equilibrium condition by $[\mathrm{H^+}][\mathrm{PO_4^-}]/[\mathrm{PO_4H}] = K_a$. The fraction of dissociated sites α, defined by $[\mathrm{PO_4^-}]/([\mathrm{PO_4^-}] + [\mathrm{PO_4H}])$, is related to K_a by the following:

$$\alpha = \frac{1}{1 + 10^{(\mathrm{p}K_\mathrm{a} - \mathrm{pH})} e^{-e\psi(0)/k_B T}} \tag{4.13}$$

with pK_a = 2.1 for the phosphate head group. This result is derived by noting that the K_a and α are given by values at $z = 0$, but $\mathrm{pH} = -\log[\mathrm{H}^+]_\infty$, where the z = 0 value is given by the following:

$$[\mathrm{H}^+] \equiv [\mathrm{H}^+]_{z=0} = [\mathrm{H}^+]_\infty \exp[-e\psi(0)/k_B T] \tag{4.14}$$

As a result, the self-consistent boundary condition is given by the following:

$$\sinh \frac{e\psi(0)}{2k_B T} = -\frac{\alpha e^2 \lambda_D}{2\epsilon_0 \epsilon_r k_B T A} = -\frac{e^2 \lambda_D}{2\epsilon_0 \epsilon_r k_B T A} \frac{1}{1 + 10^{(\mathrm{p}K_\mathrm{a} - \mathrm{pH})} e^{-e\psi(0)/k_B T}} \tag{4.15}$$

Numerical solution of $\psi(0)$ from Eq. S14 yields α for any given pH and A. As a result, α is essentially constant at 0.059 for pH values between 2.5 and 6.5 for our experimental conditions (*A* ≈ 45Å^2, T = 301 K, and $\epsilon_r = 80$).Therefore, only about 6%of the head groups are deprotonated in the absence of metal ions; however, α could be much larger in the presence of other counterions, monovalent or higher valency.

4.6.6 Literature Values for *κ*, *λ*, and *C*$_{sp}$

Values of spontaneous curvature for a variety of biological lipids that are similar in structure to the DHDP extractant were taken from Kollmitzer et al. (6). Values of bending rigidity and line tension were estimated from the literature values listed in Tables S1 and S2.

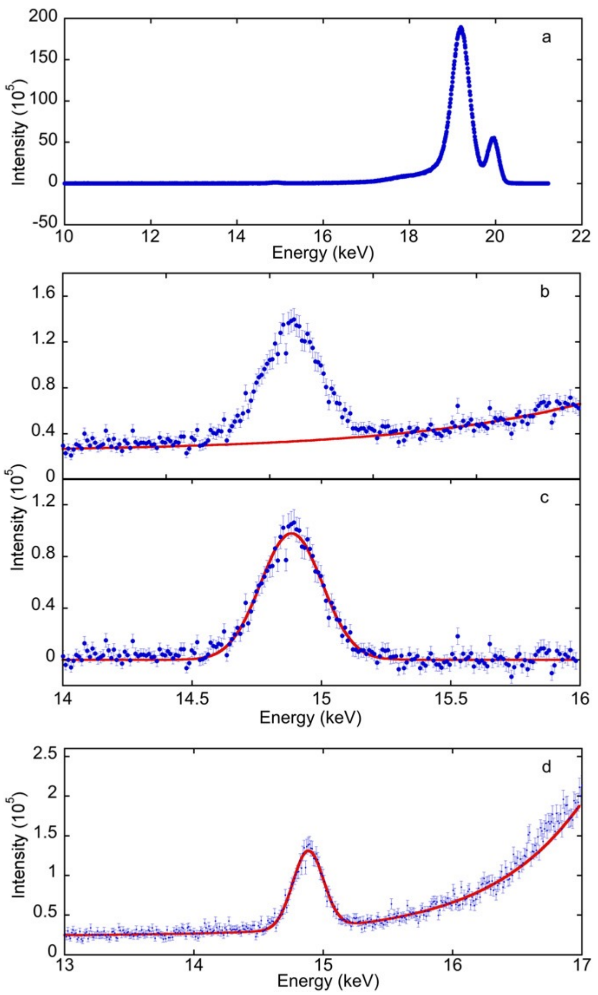


Figure 27 (A) Example of fluorescence data with the Y Kα X-ray emission line at ~14.9 keV. (B) Background fitting with an exponential decay function. (C) Peak fitting after background subtraction. (D) Fit with Gaussian peak and exponential background. Error bars represent ±1 SD.

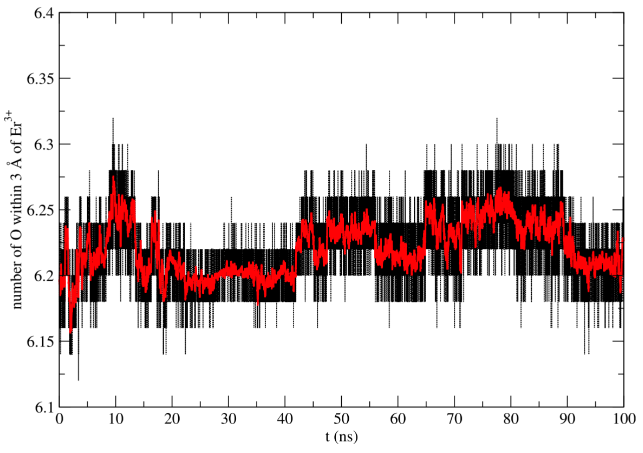


Figure 28. Number of oxygen atoms within 3 Å of the Er ion. The red line represents a 100-ps sliding average.

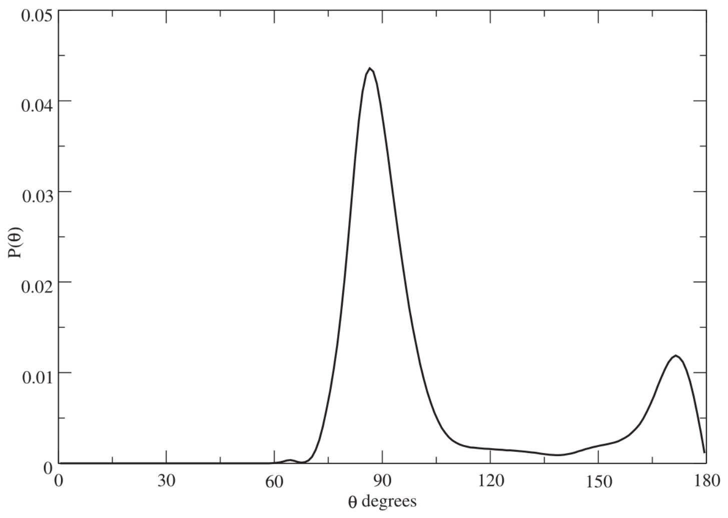


Figure 29. Distribution of O–Er–O angles for the closest six O atoms surrounding each Er ion, averaged over the 100-ns length of the trajectory. Atoms surrounding each ion were identified by first calculating all Er–atom distances that are closer than 8 Å, and then selecting the closest six (which were always oxygen atoms).

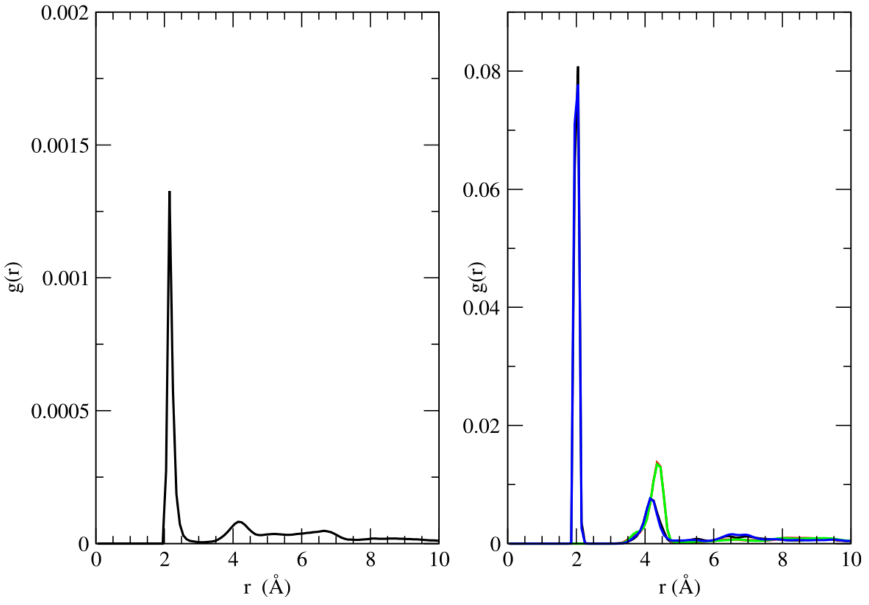


Figure 30. Radial distribution functions g(r) between (Left) Er and water oxygen, (Right) Er and the four oxygen atoms bound to the DHDP phosphorus, where the black line represents the P–O oxygen, the blue line represents the P=O oxygen, and the red and green lines represent the two ester oxygen atoms.

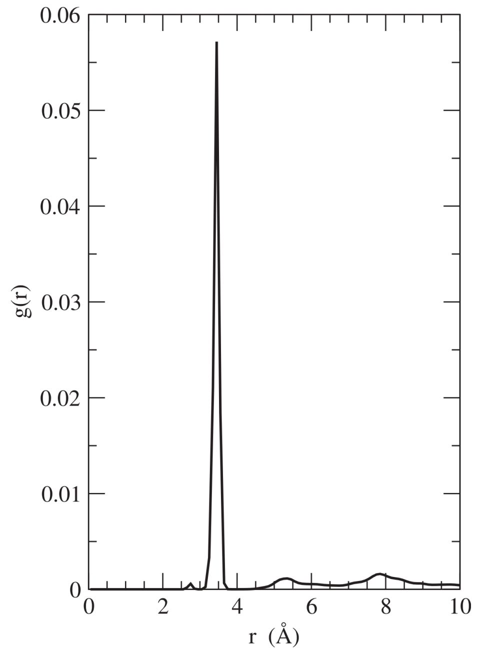


Figure 31. Radial distribution function g(r) between Er and the DHDP phosphorus

Table V Bending rigidity κ from the literature

Material	*T*, °C	κ,* k_BT	Ref.
AOT	~23	3.80	1
		1.65	2
12:0 PC	30	6.57	3
14:0 PC	24	10.45	4
	28	6.95	4
	35	7.65	4
	45	6.95	4
	60	4.10	4
14:0 PC	24	5.24	5
	25	6.44	5
	26	7.26	5
	27	7.84	5
	28	7.82	5
	30	8.24	5
	35	8.11	5
	40	7.20	5
16:0 PC	41	18.05	4
	60	4.75	4
18:0 PC	60	6.8	4
14:1 PC	30	5.38	4
16:1 PC	30	4.13	4
18:1 PC	60	4.46	4
20:1 PC	60	7.94	4

*Values in this table are one-half the values reported for bilayers.

1. Farago B, Richter D, Huang JS, Safran SA, Milner ST (1990) Shape and size fluctuations of microemulsion droplets: The role of cosurfactant. Phys Rev Lett 65:3348–3351.
2. Langevin D, Meunier J (1994) Interfacial tension: Theory and experiment. Micelles, Membranes, Microemulsions, and Monolayers, eds Gelbart WM, Ben-Shaul A, Roux D (Springer, New York), pp 485–520.
3. Kucerka N, et al. (2005) Structure of fully hydrated fluid phase DMPC and DLPC lipid bilayers using X-ray scattering from oriented multilamellar arrays and from unilamellar vesicles. Biophys J 88:2626–2637.
4. Yi Z, Nagao M, Bossev DP (2009) Bending elasticity of saturated and monounsaturated phospholipid membranes studied by the neutron spin echo technique. J Phys Cond Matt 21: 155104.
5. Chu N, Kucerka N, Liu Y, Tristram-Nagle S, Nagle JF (2005) Anomalous swelling of lipid bilayer stacks is caused by softening of the bending modulus. Phys Rev E 71:041904.

Table VI Line tension λ from the literature

Interface	λ, pN	Domain constituents	Proportions	T, °C	Ref.
Water–air	5.22 ± 1	Methyl octadecanoate	Neat	35	1
Water–air	5	DMPC-cholesterol-(dye)	68:30 mol%	~25	2
Water/glycerol/glucose–air	1.1 ± 0.3	Poly(dimethyl)siloxane	Neat	22	3
Water–air	1.6 ± 0.4	DMPC-cholesterol	70:30 mol%	~23	4
Water/AOT–air	1.37 ± 0.25	Polydimethylsiloxane	Neat	~23	5
Water–air	1.1 ± 0.3	Polydimethylsiloxane	Neat	~23	5
Water–air	7.5	Methyl octadecanoate	Neat	27.5	6
Water–air	29 ± 6	Methyl octadecanoate	Neat	~35	6
Water–hexane	30	$F(CF_2)_{10}(CH_2)_2OH$	Neat	37–40	7

1. Steffen P, Wurlitzer S, Fischer TM (2001) Hydrodynamics of shape relaxation in viscous Langmuir monolayer domains. J Phys Chem A 105:8281–8283.

2. Seul M (1993) Dynamics of domain shape relaxation in Langmuir films. J Phys Chem 97:2941–2945.

3. Mann EK, Hénon S, Langevin D, Meunier J, Léger L (1995) Hydrodynamics of domain relaxation in a polymer monolayer. Phys Rev E Stat Phys Plasmas Fluids Relat Interdiscip Topics 51: 5708–5720.

4. Benvegnu DJ, McConnell HM (1992) Line tension between liquid domains in lipid monolayers. J Phys Chem 96:6820–6824.

5. Mann EK, Primak SV (1999) Stability of two-dimensional foams in Langmuir monolayers. Phys Rev Lett 83:5397–5400.

6. Wurlitzer S, Steffen P, Wurlitzer M, Khattari Z, Fischer TM (2000) Line tension in Langmuir monolayers probed by point forces. J Phys Chem 113:3822–3828.

7. Li M, Tikhonov A, Schlossman ML (2002) An x-ray diffuse scattering study of domains in F(CF2)10(CH2)2OH monolayers at the hexane-water interface. Europhys Lett 58:80–86.

5 PROBING THE INTERFACIAL ION DISTRIBUTION IN ALSEP BACK-EXTRACTION USING X-RAY FLUORESCENCE

5.1 Abstract

Metal ions are transferred between aqueous and organic phases during solvent extraction for the purpose of separating target ions from other types of ions and ultimately isolating them in an aqueous phase. The liquid-liquid interface is believed to be the site where complexation of ions changes substantially as the ion passes from the aqueous to the organic side of the interface or vice versa. One of the challenges in the experimental study of these interfacial processes is the measurement of the interfacial density of ions independent of their presence in the nearby bulk aqueous and organic phases. Here, we demonstrate advances in the measurement and analysis of X-ray fluorescence near total reflection (XFNTR) in the back-extraction of Eu ions from an organic phase to an aqueous phase. XFNTR measures both the ion concentration in the two neighboring bulk phases and its interfacial density. Liquid-liquid interfaces studied by XFNTR are formed by placing *n*-dodecane solutions of bis(2-ethylhexyl-phosphoric) acid loaded with trivalent europium placed in contact with either pure water, nitric acid, or citric acid solutions. In addition, ICP-MS analysis of the bulk solutions provided a test of the XFNTR. These measurements detected Eu ions at interfaces with nitric and citric acid solutions, though a larger density was measured at the interface with nitric acid solutions. Europium was not present at the interface with pure water. These demonstrated capabilities of XFNTR are relevant for further *in situ* investigations of the role of the liquid-liquid interface in solvent back-extraction within a variety of processes including the ALSEP process that has been proposed for the separation of minor actinides from lanthanides.

5.2 Introduction

Used nuclear fuel contains the radioactive by-products of nuclear fission, including long-lived lanthanides and actinides. Certain actinides retain high radiotoxicity for hundreds of thousands of years, posing a challenge for their safe disposal. The separation of long-lived trivalent actinides from lanthanides is a key step for closing the nuclear fuel cycle but remains an open issue due to their similar radii, charge, and bond character. Several separation processes have been developed, which rely upon solvent extraction as the primary technology for large scale hydrometallurgical reprocessing of used nuclear fuel. These processes include Transuranic Extraction (TUREX) and Trivalent Actinide-Lanthanide Separations by Phosphorus-reagent Extraction from Aqueous Complexes (TALSPEAK).[16–18] Recycling lanthanides and actinides from used nuclear fuel not only reduces the required capacity of geological depositories for used nuclear fuel,[19,20] but offers other substantial benefits, including the recovery and reuse of the nuclear fuel.

Solvent extraction is a scalable separations technique based upon the distribution of chemical species between two immiscible liquids, usually an aqueous phase in contact with an organic phase. The extraction cycle consists of both forward- and back-extraction processes. In forward extraction, a target metal ion contained within a complex chemical mixture in an aqueous environment is extracted into an organic phase by forming a complex with an organic soluble extractant. The subsequent back-extraction process, also known as metal stripping, involves contact of the metal-loaded organic phase with a fresh aqueous solution to transfer the metal ions to the aqueous solution for further processing. The intended result of running the two processes sequentially is to separate the target metal ions from many other types of metal ions and isolate them in an aqueous phase containing only the target metal ions, along with aqueous complexants and acid buffers that may have been used to optimize the back-extraction process.

The separation of actinide and lanthanide metal ions with solvent extraction processes can be based upon their relative affinity for organic-soluble cation exchange extractants such as HDEHP (bis(2-ethylhexyl-phosphoric) acid) and aqueous complexants such as DTPA (diethylene triamine pentaacetate). While the extractants exhibit a strong affinity for both lanthanides and actinides, aqueous complexants are actinide-selective.[21] In its simplest mode of forward extraction from the water to the organic phase, each trivalent metal ion will bind six HDEHP molecules.[22] Complicating the separation of actinides from lanthanides, HDEHP co-extracts both of them. The distribution ratio significantly overlaps those of the actinides,[16] indicating the co-extraction of both lanthanides and actinides with HDEHP. One way to increase their discrimination is to use an aqueous solution of complexants to retain actinides in the aqueous phase. However, a back-extraction stage may be necessary to further strip actinides from the co-extracted mixture.

The recent development of ALSEP (Actinide Lanthanide Separation) addresses the challenge of optimizing the selective separation of minor actinides (Am, Cm) from lanthanides.[20,23] The process was designed to simplify separation steps and minimize chemical consumption and waste, hence optimizing cost and scalability to the industrial level.[19] ALSEP utilizes a combination of solvating extractants (e.g. TODGAN,N,N′,N′-Tetraoctyl diglycolamide or TODGA), cation exchange extractants (e.g. Bis(2-ethylhexyl) phosphate or HDEHP) dissolved in the aliphatic organic phase (e.g., *n*-dodecane), and aqueous complexants. The process involves both forward and back extraction. In the forward extraction stage, lanthanide and actinide metal ions are co-extracted from an acidic aqueous solution into an organic solution containing the mixture of neutral and acidic extractants. In the subsequent back-extraction, the minor actinides are selectively stripped into an acidic aqueous solution of complexant such as DTPA.[23]

Characterization of the ALSEP process has included measurements of distribution ratios and separation factors in chemical systems that utilize different choices of the neutral/acidic extractant pair, the complexant, and the acidity of the aqueous phase.[20,23] Related to these studies are measurements of x-ray absorption spectroscopy that reveal the competitive binding of lanthanide and actinide ions to neutral and acidic extractants within bulk liquid phases.[22,24–26] Despite these studies, optimizing the kinetics and efficiency of back-extraction in the ALSEP fuel processing cycles remains a challenge. The major complexants (DTPA or HEDTA) have exhibited slow complexing effects across the lanthanide series and with americium. We hypothesize that the interaction of aqueous complexants and buffers with ion-extractant complexes at the organic-aqueous interface plays a significant role in the mechanism of the back-extraction of metal ions, and that knowledge of the interfacial complexes can lead to optimization of the extraction selectivity and kinetics in the ALSEP process.

Currently, no consensus exists on the interfacial metal ion concentration or the molecular species and complexes present at the interface during the back-extraction processes. Investigations of the nanoscale structure of the organic-aqueous interface under conditions relevant to ALSEP back-extraction are needed to address these issues. Recently, we have used synchrotron X-ray reflectivity and interface-sensitive X-ray fluorescence to characterize ion-extractant complexes at the organic-aqueous interface in model systems for forward extraction, including the extraction of Er(III), Y(III), and Sr(II) ions by organo-phosphorus acidic extractants.[27–31] These results demonstrated that ion-extractant complexes are formed directly at the interface, and subsequently transported into the bulk organic phase.

In this paper, we report the first x-ray study of the organic-aqueous interface under chemical conditions that are relevant for understanding back-extraction in the ALSEP process. In this first stage of a larger project we address the need to characterize the surface density of metal ions at the organic-aqueous interface under conditions that include metal ion concentrations of roughly 1 mM in both the organic and aqueous bulk phases on either side of the interface. Here, we modify the measurement and analysis of x-ray fluorescence near total reflection (XFNTR), and apply these modifications to a spare model system of back-extraction that is relevant to the ALSEP process. In this spare system, we utilize an *n*-dodecane phase of a single extractant, HDEHP, and a single lanthanide, Eu(III), and measure the back-extraction of Eu into aqueous solutions containing either pure water, nitric acid, or citric acid. Subsequent work is planned to include more chemical components in the model system to increase the relevance of x-ray measurements of liquid-liquid interfaces to understanding and optimizing ALSEP processes.

5.3 Materials and Methods

5.3.1 Solution Preparation

The organic solvent *n*-dodecane ($CH_3(CH_2)_{10}CH_3$, >99%) was purchased from Alfa-Aesar and purified by passing it six times through a chromatography column containing activated alumina,[32] then subsequently filtered through Omnipore filter paper. Ultrapure water was produced either from a Nanopure UV Barnstead system or a Millipore Milli-Q systems and was used for all aqueous solutions. Bis(2-ethylhexyl)-phosphoric acid (HDEHP, >99.9%) was purchased from Alfa-Aesar purified via a third-phase formation procedure[33] and preloaded with Europium. Sample concentrations and pH values indicated in Table 7 were chosen to mimic values that might be used in ALSEP process.

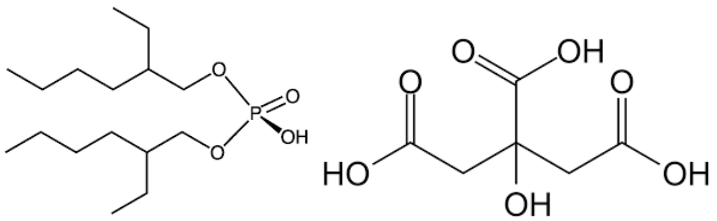


Figure 32. Molecular structure of (a) HDEHP and (b) citric acid.

Organic phases are diluted from a stock solution made by extracting Eu cations from aqueous solution into an *n*-dodecane solution of HDEHP. The feed water phase for the extraction is 100 mM $Eu(NO_3)_3$ solution. Four millimoles of $Eu(NO_3)_3•5H_2O$ was dissolved into 40 mL of water and 24.8 mmol of pure acetic acid (>99.99%, Alfa-Aesar) added as buffer. The value of pH was adjusted to 4.7 by titrating 1M NaOH solution (NaOH, 99.8%, Fisher), to ensure that the pH remains above 3 throughout the extraction, as required for full extraction of europium into organic phase.[16] The organic phase for the extraction is 0.1 M of HDEHP in *n*-dodecane. HDEHP is obtained from Alfa-Aesar (97%) and purified via a third-phase formation procedure. [33] Dodecane purchased from Sigma-Aldrich (>99%) is further purified by passing it six times through activated alumina in a chromatography column. The two phases were shaken and separated in a separation funnel, and the oil phase was taken out and centrifuged for 10 min before transferred to a glass bottle. Then 10g of anhydrous $NaSO_4$ was added in the glass bottle and left over night to dry out the remaining water in the oil phase. The final oil stock solution is obtained by filtering out the $NaSO_4$ with a membrane filter.

A concentrated stock solution (0.5 M) of $Eu(NO_3)_3$ with neutral pH was made with europium nitrate ($Eu(NO_3)_3•5H_2O$, 99.9%, Aldrich) without further purification. Nitrate solutions (1 mM) and citrate solutions (0.5 M) were made with nitric acid (HNO_3, 70% concentration, >99.999% trace metal basis, Aldrich) and citric acid ($C_6H_8O_7$, anhydrous, >99.9%, Aldrich), respectively. All water solutions were filtered through Omnipore membrane paper. Ammonium hydroxide (NH_4OH, certified ACS plus, Fisher) was used to adjust the acidity of citric acid to pH 3.

Table 7. Typical sample compositions in the experiments

Samples	Organic Phase		Aqueous phase	pH
1	1 mM Eu	10 mM HDEHP	Pure water	Neutral
2	1 mM Eu	10 mM HDEHP	1 mM nitric acid	3
3	1 mM Eu	10 mM HDEHP	0.5 M citric acid	3

5.3.2 ICP Measurements

Aliquots from the aqueous and organic phase liquids were removed from the X-ray sample cell for all the samples after X-ray measurements. They were stored in 2 mL glass vials for further analysis by ICP-MS. Vials containing aqueous phase were sent to Galbraith Laboratory Inc. Organic phases were placed in contact with 5M HNO_3 at 20:1 aqueous: organic ratio to extract all the Europium in organic phase into aqueous phase,[34] which was then analyzed by ICP-MS at UNLV.

5.3.3 Measurement Methods

X-ray fluorescence near total reflection (XFNTR) measurements from liquid-liquid interfaces were performed at ChemMatCARS, Sector 15 of the Advanced Photon Source. Dodecane-water interfaces (2.2:1 volume ratio of the bulk phases) were prepared in a custom sample cell described elsewhere.[27,30] All measurements were performed at room temperature. A beam of monochromatic X-rays of wavelength λ_x = 0.61992 Å passed through the organic phase, which lies above the water, and reflected from the dodecane-water interface. X-ray fluorescence from the sample passed through a Kapton film window placed roughly 0.5 mm above the interface and was then recorded by a Vortex-EX multi-cathode x-ray detector (SII Nano Technology USA, Inc.). Fluorescence intensity was normalized to the incident intensity measured before x-rays entered the sample cell.

XFNTR data was measured in two different ways. A so-called Q-scan measures the fluorescence intensity as a function of wave vector transfer normal to the interface $Q_z = (4\pi/\lambda_x)\sin\alpha$, where α is incident angle measured from the interfacial plane. A so-called sh-scan, on the other hand, measures the fluorescence intensity as a function of sample height sh, measured along the direction normal to the interface, at a fixed incident wave vector transfer below (Q_z = 0.006 Å^{-1}) or above (Q_z = 0.015 Å^{-1}) the critical wave vector transfer for total reflection (0.010 ~ 0.011 Å^{-1}).

Samples were equilibrated before XFNTR measurements. Although the back-extraction process begins immediately after the organic phase is placed in contact with the water phase, XFNTR measurements did not usually start until about two hours later because of the time required for sample alignment. Besides, the bulk aqueous and organic phases were gently stirred for 15 to 20 minutes. The aqueous phase was stirred by a Teflon-coated stir bar resting on the bottom of the glass try and the organic phase was stirred by two rotating propellers inserted from above. A set of test data have shown that stirring helps the sample reach a steady state (see sample 9 in Appendix D.1.1)

5.3.4 X-ray fluorescence Near Total Reflection (XFNTR) Technique

XFNTR is used to measure the distribution of Eu throughout our samples by measuring the Eu interfacial number density (per nm^2) and its concentration in each of the neighboring bulk phases. First introduced by Bloch and Yun, the elemental selectivity and surface sensitivity of XFNTR has been used to detect the existence and coverage of metal ions at liquid-vapor interfaces,[35] [31,36–38] with a more recent application to the study of solvent extraction processes at liquid-liquid interfaces.[27,28,30] Despite successful studies of the liquid-liquid interface, the application of XFNTR has been limited to studying interfacial ion densities when the aqueous bulk concentration of ions is small enough to produce negligible fluorescence and the organic bulk concentration of ions is zero. However, during the process of solvent back-extraction, ions may have a substantial concentration in both bulk phases, as well as at the interface. Here, we discuss modifications to the measurement and analysis of XFNTR that allows us to measure the concentration of ions in both bulk phases, as well as the interfacial density of ions.

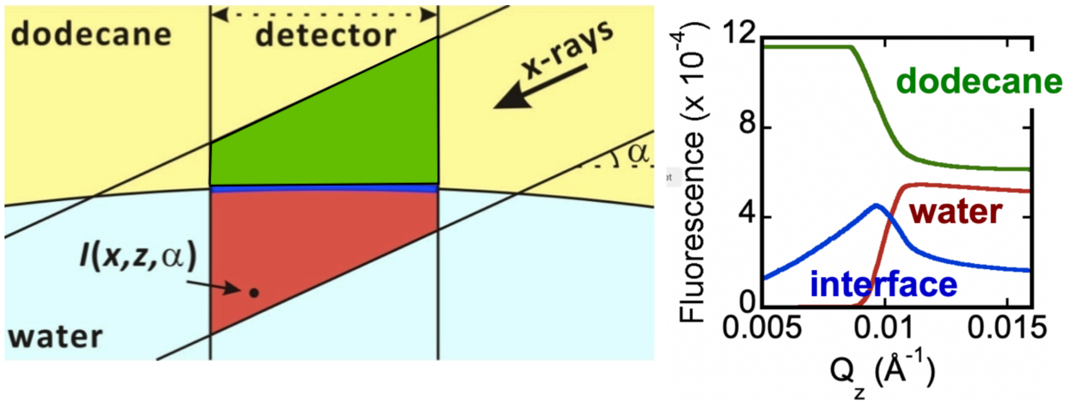


Figure 33 (a) Schematic of liquid-liquid interface with incident X-ray beam. (b) Illustration of the form of the three terms in Equation (5.1) that represent fluorescence emitted by ions in three different regions: bulk dodecane phase , bulk water phase and the interface between them.

To account for the fluorescence generated throughout the entire sample we follow the formalism in Bu et al. [27] to write the fluorescence intensity as a function of wave vector transfer Q_z as

$$I(Q_z) = C\left[n_{org}\iint I(x,z,Q_z)dxdz + n_{aq}\iint I(x,z,Q_z)dxdz + \sigma_{int}\int I(x,0,Q_z)dx\right] + I_{bg} \quad (5.1)$$

where n_{aq}, n_{org} and σ_{int} are the volume density of ions in the aqueous and organic bulk phases, and the surface density of ions at the interface, respectively. The three terms represent symbolically the X-ray fluorescence generated by metal ions in the three regions illustrated in Figure 33 (a). The detector records X-ray fluorescence emitted by ions located in the region spanned by the two vertical lines in the figure, known as the detection volume. The background term I_{bg} is generated by secondary scattering from the top phase, in which an X-ray elastically scattering from the organic phase into the aqueous phases produces fluorescence metal ions in the aqueous phase. This background contribution can contribute substantially to the signal from the aqueous phase at low Q_z.The green, dark blue and red areas in the figure represent the region of overlap between the detection volume and the incident X-rays within, respectively, the bulk dodecane phase, the interface, and the bulk aqueous phase. This formalism lacks dependence on the y-direction because the size of the footprint in that direction, which is given by the beam width of roughly 200 μm, is small compared with the detector region (12.7 mm) and the interfacial curvature (~100 m). Thus the *y* dependence is considered a constant and accounted for by the calibration factor *C*.

X-ray fluorescence from the three regions have a different dependence on the normal component of the wave vector transfer $Q_z = 4\pi/\lambda \sin\alpha$ where α is the incident angle of the incoming X-rays shown in Figure 33.a. Fluorescence from the organic phase includes fluorescence generated by ions within the path of the incident and reflected beams. Below the critical wave vector for total reflection (i.e., below a critical angle of incidence), the two components are nearly the same, whereas above the critical wave vector the reflected intensity is reduced rapidly to zero with increasing incident angle. These effects produce a roughly 2:1 ratio of the fluorescent intensity as the wave vector varies from below to above the condition for total reflection, as illustrated by the green curve in Figure 33. Fluorescence from the lower, aqueous phase occurs when X-rays penetrate the lower phase, which occurs either as an evanescent wave for wave vector transfers below that of total reflection or as a transmitted wave for larger wave vector transfers. Assuming a uniform distribution of ions in the interfacial region and the bulk aqueous phase, the penetration depth varies from roughly 10 nm well below the critical wave vector transfer to a maximum of several micrometers above the critical value over the experimental range of Q_z (0.005

Å^{-1} $\leq Q_z \leq$ 0.016 Å^{-1}). This variation in X-ray penetration depth into the aqueous phase produces a significant rise in the fluorescence signal above the critical wave vector transfer of 0.0103 Å^{-1} shown by the red curve in Figure 33. Fluorescence intensity due to the enrichment of metal ions at the interface will be dominated by the Fresnel transmissivity since the penetration depth is negligible, giving rise to the shape of the blue curve in Figure 33.[39]

The fluorescence intensity is calculated from Equation (5.1) by integrating the X-ray intensity weighted by the concentration of metal ions along the X-ray pathways illustrated in Figure 34. Only the part of the x-ray paths that falls within the detection volume, i.e., the two vertical lines, produces fluorescence measured by the detector. Figure 34 illustrates an X-ray beam striking the sample interface with incident angle α. The intensity of the incident beam is measured to have a nearly gaussian variation with standard deviation $\sigma_D = 4.6 \pm 0.01$ μm (FWHM $=$ 10.85 ± 0.16 μm). Calculations of the fluorescence utilize a beam height $h = 6\sigma_D$ to account for 99.7% of the beam intensity. X-rays strike a slightly curved dodecane-water interface whose curvature is modeled by a constant radius of curvature R_C that is measured by x-rays to be typically 100 m.[10] Curvature of the interface in the transverse direction (the y-direction) can be neglected because the dimension of the x-ray footprint (on the interface) in this direction is small (200 μm). The dimension of the footprint along the x-direction, h/α (for small α), is larger than the length l (= 12.7 mm) of the detection volume for the range of measured values of Q_z. These values span the critical Q_c, which varies from 0.0103 Å^{-1} to 0.0112 Å^{-1}, depending upon the chemical composition of the phases.

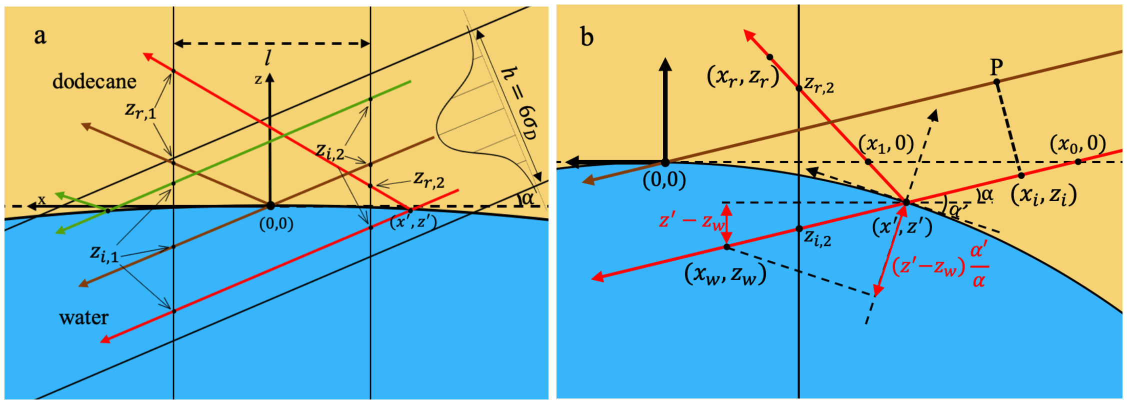


Figure 34. (a) Schematic diagram of X-ray paths used for the calculation of the fluorescence intensity. Red, brown and green rays strike the interfaces along the *x*-direction at values $x' < -l/2$, $-l/2 < x' < l/2$ and $x' > l/2$, respectively where x' indicates the position on the curved interface and the *x*-axis is positive to the left of center. (b) detailed view of the geometry around the position where red ray strikes the interface. Note that $x' < 0$ in this region. The incident angle and interfacial curvature are exaggerated for clarity.

Rays within the X-ray beam are shown to be parallel in Figure 34 because the angles of incidence α are roughly 100 times larger than the beam divergence. Figure 34 illustrates three rays colored red, brown and green, to represent rays that strike the surface beyond (red and green) and within (brown) the detector region. The red ray shown in Figure 34(b) passes through position (x_i, z_i) in the organic phase and strikes the interface at (x', z') with incident angle $\alpha' \approx \alpha - x'/R_C$ (for $R_C \gg h/\alpha$). The transmitted ray passes through (x_w, z_w) in aqueous phase and the reflected ray passes through (x_r, z_r) in the organic phase. The brown ray is chosen to pass through the origin $(0, 0)$ for the convenience of normalization. The dashed line connecting (x_i, z_i) on red ray and point P on brown ray is perpendicular to both rays, hence the X-ray intensity at (x_i, z_i) on the red ray is given by

$$I_{org}^{inc}(x_i, z_i) \approx I(0,0) \cdot e^{-\mu_{i,o}(x' - \frac{z_i - z'}{\alpha})}, x' \in [-l/2, f/2] \tag{5.2}$$

where the approximations are made to $O(\alpha^2)$ for the range of nominal angles of incidence α $2.5 \times 10^{-4} \leq \alpha \leq 7.9 \times 10^{-4}$ over the range of measured values of Q_z; $\mu_{i,o}$ is the absorption coefficient of the incident beam in the organic phase (0.273 cm^{-1} at 20 keV)[40] that determines the exponential decay in X-ray intensity with distance along the beam path; *f* is the footprint on the interface of the X-ray beam of height $h = 6\sigma_D$. Similarly, the intensity for the reflected and transmitted at positions (x_r, z_r) and (x_w, z_w), respectively, are given by

$$I_{org}^{ref}(x_r, z_r) \approx I(0,0) \cdot R(\alpha')e^{-\mu_{i,o}(x' + \frac{z_r - z'}{\alpha - 2x'/R_C})}, x' \in [-f/2, l/2] \tag{5.3}$$

and the intensity of the transmitted ray at position (x_w, z_w) is given by

$$I_{aq}^{inc}(x_w, z_w) \approx I(0,0) \cdot T(\alpha')e^{-\frac{\alpha'}{\alpha}(z' - z_w)\frac{1}{\Lambda(\alpha')}}e^{-\mu_{i,o}x'}, x' \in [-f/2, l/2] \tag{5.4}$$

where $R(\alpha') = \left|\left(\alpha' - \sqrt{\alpha'^2 - \alpha_c^2}\right)/\left(\alpha' + \sqrt{\alpha'^2 - \alpha_c^2}\right)\right|^2$ is the Frenel reflectivity and $T(\alpha') = \left|2\alpha'/\left(\alpha' + \sqrt{\alpha'^2 - \alpha_c^2}\right)\right|^2$ is the Fresnel transmissivity (where $\alpha_c = Q_c/2k_0$ is the critical angle); the radius of curvature R_C is measured for some samples and used as a fitting parameter for others;[10] $\Lambda(\alpha') = 1/\left[2k_0 \mathrm{Im}\left(\sqrt{\alpha'^2 - \alpha_c^2 + 2i\beta}\right)\right]$ is the X-ray penetration depth that determines the exponential decay in X-ray intensity along the direction perpendicular to the surface (where β is the imaginary part of the refractive index of the aqueous phase). Note that for the red ray in Figure 34(b), the depth in the aqueous phase is $\alpha'(z' - z_w)/\alpha$, in the presence of curvature, instead of the apparent depth $z' - z_w$.

The emitted X-ray fluorescence undergoes an additional absorption along the path from its point of emission to the detector. Recalling that all intensities are scaled to a normalized intensity at $I(0,0)$, an absorption factor $e^{\mu_e z}$ is needed for the energy of the fluorescence emission. Finally, the fluorescent intensity is written under that assumption that the metal ion distribution in the organic and aqueous phases is uniform, except along the *z*-direction within the interfacial region. Therefore, the fluorescence intensity produced by metal ions at a certain position is proportional to the incident X-ray intensity and ion number density. Integrating the above equations along the z direction yields the total fluorescence produced by the ray that strikes at (x', z'),

$$\begin{aligned}\frac{I_{flu}(x',\mathrm{z}')}{I(0,0)} &= n_{oil}\left(\int \frac{I_{oil}^{inc}}{I(0,0)}\,\mathrm{e}^{\mu_{e,o}z_i}\mathrm{d}z_i + \int \frac{I_{oil}^{ref}}{I(0,0)}\,\mathrm{e}^{\mu_{e,o}z_r}\,\mathrm{d}z_r\right) \\ &\quad + n_{water}\int \frac{I_{aq}^{inc}}{I(0,0)}\,\mathrm{e}^{\mu_{e,w}z_w}\mathrm{d}z_w + \sigma_{int}\left.\frac{I_{aq}^{inc}}{I(0,0)}\right|_{z_w=z'}\end{aligned} \tag{5.5}$$

where n_{org} and n_{water} , and $\mu_{e,o}$ and $\mu_{e,w}$, are the ion number densities, and fluorescence emission line absorption coefficients, for the organic and aqueous phases, respectively, where values of the latter are $\mu_{e,w}$= 26.58 cm^{-1} and $\mu_{e,o}$= 7.451 cm^{-1} for the Eu $L\alpha$ line at 5.842 keV.[40] Fluorescence intensity that originates from the interfacial region ($0 < |[(z' - z_w)\alpha'/\alpha]| < \Lambda(\alpha')$) is not negligible if there is an interfacial excess of metal ions with density σ_{int} (*i.e.,* per area), which can be considered to sufficient accuracy by setting $z_w = 0$. All the integration results up to this point can be written in analytical form. Also note that whether a term in equation (5.5) contributes to the measured fluorescence depends on the position the ray hits the interface. For example, the brown ray contributes all the terms, while the red ray contributes reflected and transmitted intensity only, and for the green ray the only non-zero component is the incident intensity.

Integrating each term in equation (5.5) involves careful consideration of the x-ray paths. For the incident component, and recalling that $f > l$ in our experiments, the first term in equation (5.5) can be written as,

$$\int \frac{I_{org}^{inc}}{I(0,0)} e^{\mu_{e,o} z_i} dz_i = \begin{cases} 0 & -f/2 < x' < -l/2 \\ \int_{z'}^{z_{i,2}} e^{-\mu_{i,o} x_0}\, e^{\mu_{e,i} z_i} dz_i & -l/2 < x' < l/2 \\ \int_{z_{i,1}}^{z_{i,2}} e^{-\mu_{i,o} x_0}\, e^{\mu_{e,i} z_i} dz_i & l/2 < x' < f/2 \end{cases}$$

$$= \frac{1}{\mu_{e,i}} e^{-\mu_{i,o} x_0} \begin{cases} 0, & -f/2 < x' < -l/2 \\ e^{\mu_{e,i} z_{i,2}} - e^{-\mu_{e,i} z'}, & -l/2 < x' < l/2 \\ e^{\mu_{e,i} z_{i,2}} - e^{\mu_{e,i} z_{i,1}}, & l/2 < x' < f/2 \end{cases} \tag{5.6}$$

where $x_0 = x' + z'/\alpha$ is the intersection of the incident ray with the x-axis illustrated in Figure 34(b), $\mu_{e,i} = \mu_{i,o}/\alpha + \mu_{e,o}$ is an effective absorption coefficient; $z_{i,1} = (x_0 - l/2)\alpha$ and $z_{i,2} = (x_0 + l/2)\alpha$ are the heights illustrated in Figure 34 (a) of the incident X-ray at the boundaries of the detection volume.

The geometry is similar for the reflected beam,

$$\int \frac{I_{oil}^{ref}}{I(0,0)} \mathrm{e}^{\mu_{e,o} z_r} \mathrm{d}z_r = R(\alpha') \begin{cases} \int_{z_{r,1}}^{z_{r,2}} e^{-\mu_{i,o} x_1} e^{\mu_{e,r} z_r} \mathrm{d}z_r, & -f/2 < x' < -l/2 \\ \int_{z'}^{z_{r,2}} e^{-\mu_{i,o} x_1} e^{\mu_{e,r} z_r} \mathrm{d}z_r & -l/2 < x' < l/2 \\ 0, & l/2 < x' < f/2 \end{cases}$$

$$= \frac{1}{\mu_{e,r}} e^{-\mu_{i,o} x_1} R(\alpha') \begin{cases} e^{\mu_{e,r} z_{r,2}} - e^{\mu_{e,r} z_{r,1}} & -f/2 < x' < -l/2 \\ e^{\mu_{e,r} z_{r,2}} - e^{\mu_{e,r} z'}, & -l/2 < x' < l/2 \\ 0, & l/2 < x' < f/2 \end{cases} \tag{5.7}$$

where $x_1 = x' - z'/(\alpha - 2x'/R_C)$ is the intersection of the reflected X-ray with the x-axis illustrated in Figure 34(b), $\mu_{e,r} = -\mu_{i,o}/(\alpha - 2x'/R_C) + \mu_{e,o}$ is an effective absorption coefficient; $z_{r,1} = -(x_0 + l/2)(\alpha - 2x'/R_C)$ and $z_{r,2} = -(x_0 - l/2)(\alpha - 2x'/R_C)$ are the heights illustrated in Figure 34(a) of the reflected X-ray at the boundaries of the detection volume.

The term for the aqueous phase is evaluated as follows:

$$
\begin{aligned}
&\int \frac{I_{aq}^{inc}}{I(0,0)} e^{-\mu_{e,w}|z_w|} \mathrm{d}z_w \\
&= T(\alpha') e^{-\mu_{i,o}\, x'} e^{-\frac{\alpha' z'}{\alpha \Lambda(\alpha')}} \begin{cases} \int_{z_{i,1}}^{z_{i,2}} e^{-|z_w| \widetilde{\Lambda}(\alpha')} \, \mathrm{d}z_w & -f/2 < x' < -l/2 \\ \int_{z_{i,1}}^{z'} e^{-|z_w|/\widetilde{\Lambda}(\alpha')} \, \mathrm{d}z_w & -l/2 < x' < l/2 \\ 0 & l/2 < x' < f/2 \end{cases} \\
&= \widetilde{\Lambda}(\alpha') T(\alpha') e^{-\mu_{i,o} x' - \frac{\alpha' z'}{\alpha \Lambda(\alpha')}} \begin{cases} e^{\frac{z_{i,2}}{\widetilde{\Lambda}(\alpha')}} - e^{\frac{z_{i,1}}{\widetilde{\Lambda}(\alpha')}} & -f/2 < x' < -l/2 \\ e^{\frac{z'}{\widetilde{\Lambda}(\alpha')}} - e^{\frac{z_{i,1}}{\widetilde{\Lambda}(\alpha')}} & -l/2 < x' < l/2 \\ 0 & l/2 < x' < f/2 \end{cases}
\end{aligned}
\tag{5.8}
$$

where $1/\widetilde{\Lambda}(\alpha') = (\alpha'/\alpha) \cdot 1/\Lambda(\alpha') + \mu_{e,w}$ is an effective penetration depth of the incident beam into the aqueous phase that accounts for X-ray absorption of the incident and fluorescent X-rays.

Equation (5.8) calculates the fluorescence intensity created by ions in regions where $z_w \leq z'$ which includes the entire aqueous phase, including the interfacial region. However, this calculation, by itself, does not fully account for interfacial ions because it does not account for a difference between the density of interfacial ions and those in the bulk aqueous phase. This can be seen by noting that the equation (5.5) contains a product of a constant aqueous phase density n_{water} and the fluorescence calculated by equation (5.8). This is equivalent to assuming that the constant density n_{water} is uniformly extended up to a mathematically defined interface, such as a Gibbs dividing surface. Note that this effect can be easily observed in a standard XFNTR aqueous calibration sample by noting the increase of fluorescence data below the critical Q (see Appendix D.2.2 sample 6, inset figure). This increase occurs because the evanescent wave penetration depth increases as Q varies from far below Q_c, say, 0.005 Å^{-1}, to near, but still below, say 0.01 Å^{-1}. These uniformly distributed ions account for some of the ions in the interfacial region, but may undercount them or overcount them, respectively, if there is a positive or negative surface excess of ions in the thermodynamic sense. In addition, spatial variations in the interfacial ion density with the normal coordinate z are not considered.

If there are excess metal ions at the interface, for example, in the form of ions bound to interfacial species, such as amphiphilic extractants, then the interfacial region occupied by these ions is much smaller than the evanescent wave decay length $\Lambda(\alpha')$, which has its smallest value of 10 nm at its smallest measured value of Q, 0.005 Å^{-1}.[27] Under these circumstances, the exponential decay in the z-direction vanishes from the integration and the interfacial fluorescence is obtained by setting $z_w = z'$:

$$\left.\frac{I_{aq}^{inc}}{I(0,0)}\right|_{z_w=0} = \begin{cases} T(\alpha')e^{-\mu_{i,o}x'} & -l/2 < x' < l/2 \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

However, a positive or negative excess of metal ions may exist throughout the interfacial region defined, for the purpose of these measurements, as that probed by the X-ray evanescent wave. For example, if the interface is charged or if there is an electric potential difference between the two bulk phases, then an electrical double layer may extend throughout the interfacial region. Alternatively, an interfacial structure may have nucleated at the interface, then grown throughout the interfacial region towards the bulk aqueous phase. Under these circumstances the exponential decay in the z-direction cannot be neglected when accounting for excess ions. One way to cut off the decay is to choose z_w in the range where $|(z' - z_w)\alpha'/\alpha| < 5\Lambda(\alpha')$, where we have chosen $5\Lambda(\alpha')$ because that accounts for 99.7% of the evanescent wave region. Now, the interfacial term will become

$$\int \frac{I_{aq}^{inc}}{I(0,0)} \mathrm{d}z = \begin{cases} T(\alpha')e^{-\mu_{i,o}x'} \displaystyle\int_{z'-\Delta}^{z'} e^{-\frac{\alpha'}{\alpha\Lambda(\alpha')}(z'-z_w)}\, \mathrm{d}z_w & -l/2 < x' < l/2 \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} T(\alpha')e^{-\mu_{i,o}x'}\dfrac{\alpha\Lambda(\alpha')}{\alpha'}(e^{\frac{\alpha'\Delta}{\alpha\Lambda(\alpha')}} - 1) & -l/2 < x' < l/2 \\ 0 & \text{otherwise} \end{cases}$$

where $\Delta = 5\Lambda(\alpha')\alpha/\alpha'$. Qualitatively, one expects that if the distribution of excess ions varies over the length scales that are either much smaller or much larger than the range of the evanescent wave decay lengths below Q_c that are probed by the measurement, then the variation of the fluorescent intensity with Q_z will not be sensitive to the form of the ion distribution. However, measurements of XFNTR may be sensitive to the form of the ion distribution when the distribution varies over length scales that lie within the length of evanescent decay lengths utilized by the experiment.

Finally, the total fluorescence intensity detected by the fluorescence detector can be obtained by summing the fluorescence produced by all X-rays but weighting the beam intensity to account for the beam profile.

$$\frac{I(\alpha)}{I_0} = C\int_{-f/2}^{f/2} w(x')\frac{I_{flu}(x')}{I_0}\mathrm{d}x' + I_{bg} \tag{5.10}$$

where C is the scale factor for a given emission line that accounts for the effect of the scattering geometry and $w(x')$ follows a normal distribution $N(0,(e/\alpha)^2)$ in a given range:

$$w(x') = \begin{cases} \dfrac{1}{\sqrt{2\pi}\sigma_D/\alpha}e^{-\frac{x'^2}{2\,(\sigma_D/\alpha)^2}} & -\dfrac{3\sigma_D}{\alpha} < x' < \dfrac{3\sigma_D}{\alpha} \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

The integral in Eq. (5.10) is calculated numerically.

The model described above assumes the alignment of three things: the center of detector region, the center of the interface ($x = 0$) and the center of the X-ray footprint on the interface. In practice however, even a careful alignment could produce an offset in sample height Δsh in an order of micrometers over the Q_z range of measurement. This leads to an equivalent horizontal offset of the center of the X-ray footprint by $\Delta sh/\alpha$ of several millimeters, which is significant compared to the detector range (12.7mm). Moreover, the horizontal misplacement of the detector could shift the detector region off the $x = 0$ position by as much as 2~3 mm, and this can also be viewed as an offset of X-ray footprint to the detector region. It also causes the center of the (x, z) coordinate system attached to the interface to move away its origin, which is negligible with a big curvature. As a result, equation (5.10) should be rewritten to include the offset

$$\frac{I(\alpha, \Delta sh)}{I_0} = C \int_{-\frac{\Delta sh}{\alpha}-f/2}^{-\frac{\Delta sh}{\alpha}+f/2} w(x' + \frac{\Delta sh}{\alpha}) \frac{I_{flu}(x')}{I_0} \mathrm{d}x' + I_{bg} \tag{5.12}$$

From the equation above, the measured fluorescence is a function of the offset Δsh of the sample height, as well as the incident angle α. In fact, instead of being considered as a source of error, variations in Δsh can be used as an extra dimension to study the fluorescence intensity. In other words, an sh-scan, a scan that varies Δsh at a fixed incident angle, might provide complementary information to a Q_z-scan, a scan that varies the incident angle α only.

Figure 35 shows the two different kinds of scans for samples containing citric acid as an example. We have chosen to illustrate this with citric acid samples because the fluorescence signal is nearly featureless as a result of the coincidental addition of the signals from the bulk organic and aqueous phases. Figure 33b illustrated the form of the signal from these bulk phases, which suggests that an appropriate bulk concentration of metal ion in each of the phases might produce a nearly featureless total XFNTR signal in the absence of an interfacial signal. This is illustrated by the Q_z-scan in Figure 35a for which the rise in the XFNTR signal above Q_c from the bulk water phase is similar in magnitude to the fall in the signal from the bulk organic phase. For this sample, the slight increase in XFNTR signal below Q_c indicates the presence of excess metal ions at the interface, but the interface signal is weak. The overall effect is that the measured XFNTR signal is nearly featureless and a skeptic might be concerned that something is wrong with the measurement. In cases like this, stronger signals from each of the bulk phases are evident in an XFNTR sh-scan, which can provide a confirmation of the analysis of an XFNTR Q_z-scan.

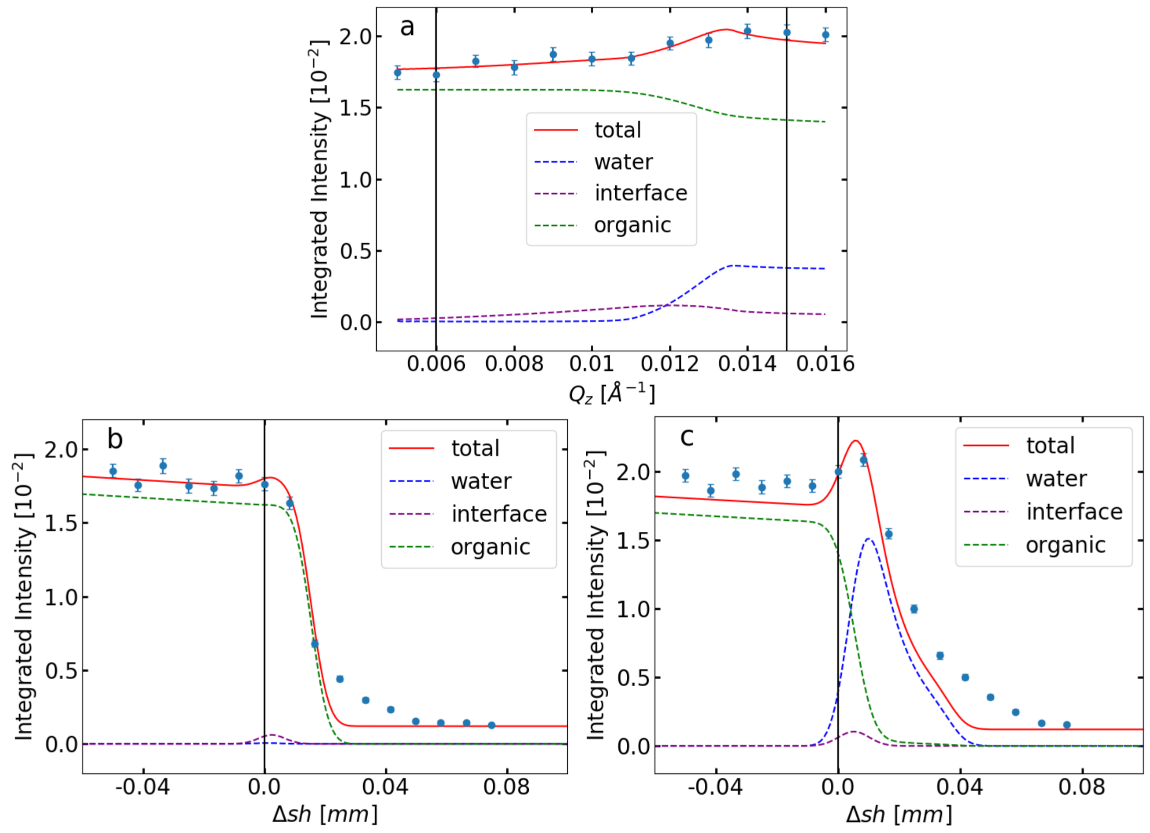


Figure 35. XFNTR the citric acid sample containing 0.5 M citric acid and 0.1 mM DTPA in the aqueous phase and 1 mM Eu 10 mM HDEHP in the oil phase. (a) Q_z-scan; (b) sh-scan at $Q_z = 0.006$ Å^{-1}; (c) sh-scan at $Q_z = 0.015$ Å^{-1}. Vertical lines mark values discussed in the text.

In the XFNTR Q_z-scan in Figure 35(a), Δsh is a fitting parameter. The fitting result yields $\Delta sh = -2$ µm, meaning there is a negative sample height offset due to the misalignment of the instrument. The two XFNTR sh-scans in (b) and (c) were performed below ($Q_z = 0.006$ Å^{-1}) and above ($Q_z = 0.015$ Å^{-1}) the critical angle, at the values indicated by the two black vertical lines in (a), respectively. For low sample height positions ($\Delta sh \ll 0$), the X-ray beam pases above the interface and penetrates through the bulk organic phase; therefore, the fluorescence intensity consists of only that created by the incident beam, which is nearly constant ($\sim 1.9 \times 10^{-2}$)

regardless of the sample height. The reflected and transmitted beam start to move into the detection region as the sample moves up, and the intensity above the critical angle consists of all the terms in Equation (5.5). At even higher sample heights, part of the beam strikes the wall of the glass tray and is partially blocked. Note that the wall, which is ~3.2 mm in thickness, does not fully prevent the X-ray from passing through the glass wall. These X-rays create fluorescence signal from the aqueous phase, and produce the shoulder in the fluorescence for Δsh = 0.03 to 0.06 mm. Our model doesn't account for this effect; therefore, the predicted fluorescence is lower than the data in this range of Δsh. There is also a universal background ($\sim 0.12 \times 10^{-2}$) that is the result of second scattering from the organic phase. The secondary scattering is the result of elastic scattering of X-rays from the organic phase that travel into the aqueous phase and create fluorescence from metal ions in the aqueous phase.

The total intensity, along with its components, as the function of Δsh is drawn in Figure 35 (b) and (c) using the number densities derived from the fit in in Figure 35(a). The intensity arising from interface is small and bears a large error bar in all three cases, suggesting the difficulty of demonstrating the existence of interfacial ions in this sample. The fluorescence from the bulk aqueous phase below critical angle is zero due to total reflection. Above the critical angle a significant part of the intensity comes from aqueous phase in the $\Delta sh > 0$ region. This contribution peaks at around $\Delta sh = 0.01$ mm before being reduced to zero as the detection volume in the aqueous phase decreases. As a result, the total intensity features a peak at the same location, which confirms the fitting result for the metal ion concentration in the aqueous phase. Therefore, the XFNTR sh-scans can provide a separation of fluorescence signals from the organic bulk phase and aqueous bulk phase signals that are otherwise summed in an XFNTR Q_z-scan. This is particularly useful when the XFNTR Q_z-scan is nearly featureless as in Figure 35a.

One can also fit the sh-scan data directly to equation (5.12) to obtain a number density in the organic and aqueous phases, as well as the interface. This suggests the possibility of measuring XFNTR in a two-dimensional grid in $(\alpha, \Delta sh)$ space, which might reduce the error bars on the fitted parameters, i.e., the bulk ion concentrations and interfacial ion density. This could be an interesting extension to the original XFNTR method.

5.4 Results and Discussion

The Europium $L\alpha$ emission line at 5.849 keV in the raw fluorescence spectrum is fitted to a Gaussian peak as shown in Figure 36(a). The product of the height and FWHM of the Lα peak is used to produce data points in Figure 36(b), which shows one example set of fluorescence measurements fitted to equation (5.12) to find n_{org}, n_{aq} and σ_{int}, hence ion concentrations, for each phase of the samples in Table 7. The averaged results of this fitting for four different samples are shown in Table 8. All of the data and results can be found in Appendix D). The X-ray result for the neutral (pure) water sample shows no ions in water phase ($0^{+0.015}_{-0.017}$ mM), a result that is confirmed by ICP-MS ($< 3 \times 10^{-5}$ mM). The interfacial distribution ($2.9^{+0.6}_{-0.5} \times 10^{-3}$ Å^{-2}) is the average of three measurements, two of which show no interfacial ions whereas the third shows an interfacial density of $8.74^{+1.68}_{-0.87} \times 10^{-3}$ Å^{-2}. The results from a sample without interfacial ions are plotted in Figure 36(b). The intensity below the critical wave vector transfer (Q_c = 0.01023 Å^{-1}) is flat, which is typical of XFNTR data when ions are present only in the dodecane phase, as illustrated in Figure 33(b). The data are slightly curved near Q_c due to the presence of a positive curvature of the interface. The ratio of intensity at $Q_z = 0.006$ Å^{-1} to that at $Q_z = 0.016$ Å^{-1} is roughly 1.88, which is lower than the ideal 2:1 ratio as the result of a small positive sample height offset in the measurement that enhances the fluorescence signal from the aqueous phase.

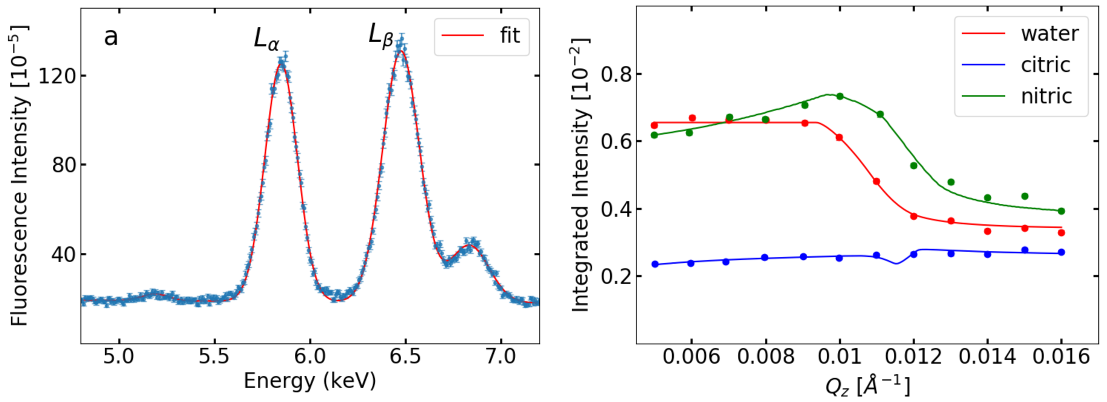


Figure 36 (a) the L_α and L_β emission line of Europium and the fitted gaussian peaks. (b) Representative X-ray fluorescence near total reflection (XFNTR) data as a function of Q_z from the liquid-liquid interface between an *n*-dodecane solution of 1 mM Eu with 10 mM HDEHP and different aqueous phases: pure water (red), 1mM HNO3 pH=3 solution (green) and 0.5M citric acid pH=3 solution (blue).

The nitrate sample was made from the same solutions as the pure water sample with the addition of 1mM HNO_3 in the aqueous phase ($\mathrm{pH} = 3 \pm 0.1$). The XFNTR data mostly follows the trend of the pure water sample, indicating a similar equilibrium dominated by Eu ions in the oil phase. The fitted concentration of Eu ions in the organic phase is $0.96^{+0.01}_{-0.01}$ mM, the ICP-MS result is 1.020 ± 0.1 mM. Moreover, the positive slope below the critical wave vector transfer for total reflection, $Q_c \approx 0.01\ Å^{-1}$, indicates the presence of an interfacial excess of ions, as illustrated by the interfacial contribution shown in Figure 33 (b). Quantitative analysis yields an Eu surface number density of $6.28^{+0.28}_{-0.24} \times 10^{-3} Å^{-2}$, or one Eu ion for every $159.2^{+7.1}_{-6.8}\ Å^2$ of the interface. The result also shows absence of metal ions ($0^{+0.022}_{-0.013}$ mM) in the water phase as confirmed by ICP-MS results ($< 3.85 \times 10^{-5}$ mM).

Table 8 Averaged X-ray results of ion distributions of four samples for three different aqueous composition: pure water, 1mM HNO3 and 0.5M Citric, compared with ICP-MS results (below X-ray restuls)

Aqueous phase	Ion Concentration (X-ray / ICP-MS)		
	Bulk Organic (mM)	Interface ($10^{-3}Å^{-2}$)	Bulk Aqueous (mM)
Water	1.05(+0.11/-0.011)	2.91(+0.60/-0.54)	0 (+0.015/-0.017)

	1.089*	N/A	$< 3 \times 10^{-5}$
Nitrate	0.961(+0.005/-0.096)	6.28(+0.26/-0.24)	0 (+0.022/-0.013)
	1.020	N/A	$< 3.85 \times 10^{-5}$
Citrate	0.443(+0.054/-0.005)	0.35(+0.16/-0.19)	0.683(+0.016/-0.016)
	0.71	N/A	0.62

* all the ICP-MS results are subject to an error bar of 10% of its measured value.

For the citrate sample, the ion concentration is $0.44^{+0.05}_{-0.01}$ mM in organic phase and $0.68^{+0.02}_{-0.02}$ mM in aqueous phase. The fluorescence data is relative featureless across the whole measuring range of Q_z. The curves for the bulk aqueous and organic phases shown in Figure 33 (b) suggest that an appropriate distribution of metal ions in the two bulk phases can lead to relatively flat variation of fluorescence with Q_z. Besides, the small positive slope below the critical wave vector transfer, $Q_c \approx 0.01$ Å^{-1} , suggests a small density of Eu at the interface ($0.35^{+0.16}_{-0.19} \times 10^{-3}$ Å^{-2}).

In equation (5.9), we assumed that all the interfacial metal ions were bound to surfactants right at the interface, or close enough to it — that is, closer than 5 to 10 nm — that $z_w = z'$ is a valid approximation. Now we discuss an alternative situation to inquire about its applicability to these experiments, namely that the concentration of metal ions is enhanced but decay along z to the value of the bulk concentration in the aqueous phase. However, during the dynamic solvent extraction, it is possible that the concentration of near interfacial ions is enhanced but decays to the bulk concentration along z direction into the aqueous phase. One possible way that this distribution could arise is if there is a difference between the bulk aqueous and organic phases. Here, we consider the resultant ion distribution that has been described by the Gouy–Chapman theory, [41,42] thought the limitations of this theory are well known. Solving the Poisson–Boltzmann equation in the presence of an infinite charged plane with positive potential $\phi(0)$ at the surface of the plane, one can obtain the near interfacial distribution in the *z* direction for ions of valence Z:

$$n_{\pm}(z) = n_{aq}\left[\frac{1 \mp e^{-z/\lambda_D}\tanh[Z_{\pm}e\phi(0)/4k_BT]}{1 \pm e^{-z/\lambda_D}\tanh[Z_{\pm}e\phi(0)/4k_BT]}\right]^2 \tag{5.13}$$

where the plus and minus signs in $n_{\pm}(z)$ represent cations and anions, respectively; n_{aq} is the aqueous bulk concentration, $Z_{\pm}e$ is the electrical charge carried by the ions and λ_D is the Debye length that measures the spatial extent of charge screening on a single ion, given below:

$$\lambda_D = \sqrt{\frac{\varepsilon_0\varepsilon_r k_B T}{e^2\sum_{i=\pm} Z_i^2 c_i^{bulk}}} \approx \frac{3}{\sqrt{c^{bulk}[\mathrm{M}]}}[\text{Å}] \tag{5.14}$$

where the approximation holds for a monovalent ($Z_{\pm} = \pm 1$) strong electrolyte. This approximation yields $\lambda_D \approx 100$ Å for a 1mM solution. Solutions of trivalent ions, containing ions such as Eu^{3+}, of concentration on the order of 1 mM, might be expected to have $\lambda_D \approx 30$ Å. Thought the applicability of this classical ion distribution theory to trivalent ions has been questioned, we anticipate that the Debye length will be an upper limit to the correct relevant length scales.

Far below the critical angle at $Q_z = 0.005\ \text{Å}^{-1}$, the evanescent wave decay length is $\Lambda(\alpha) \approx 100$ Å, and the interfacial region probed by the evanescent wave is thus roughly $5\Lambda(\alpha) \approx 500$ Å. Considering the evanescent wave region below a flat interface in which the excess metal ions follow a distribution $n(z_w)$, the excess fluorescence produced by excess ions at position (x_w, z_w) is given by rewriting equation (5.4) as follows

$$\begin{aligned}\frac{I_{eva}^{flu}(x_w, z_w)}{I(0,0)} &\approx C \cdot n(z_w) \cdot T(\alpha) e^{-\mu_{i,o}\left(x_w - \frac{z_w}{\alpha}\right)} e^{\frac{-|z_w|}{\Lambda(\alpha)}} e^{-\mu_{e,w}|z_w|} \\ &= C \cdot n(z_w) \cdot T(\alpha) e^{-\mu_{i,o}x_w} e^{-|z_w|/\tilde{\Lambda}(\alpha)}\end{aligned} \tag{5.15}$$

where $1/\widetilde{\Lambda}(\alpha) = 1/\Lambda(\alpha) + \mu_{e,w} - \mu_{i,o}/\alpha \approx 1/\Lambda(\alpha)$. The total fluorescence created by the evanescent wave region is therefore the integration of the above equation over a rectangular area below the interface. The integration area spans the entire detector region in the x direction from the interface down to $3\Lambda(\alpha)$ into the aqueous phase in the z direction:

$$\frac{I_{eva}^{flu}}{I(0,0)} = C \cdot T(\alpha) \int_{-l/2}^{l/2} e^{-\mu_{i,o} x_w}\, \mathrm{d}x_w \int_{-3\Lambda(\alpha)}^{0} n(z_w) e^{-\frac{|z_w|}{\widetilde{\Lambda}(\alpha)}} \mathrm{d}z_w \tag{5.16}$$

We then calculated the integrated fluorescence intensity for different distributions:

$$n(z_w) = \begin{cases} \sigma_{int}\delta(z_w) & \text{bound} \\ n_{aq}\left[\dfrac{1 \mp e^{-|z_w|/\lambda_D}\tanh[Z_\pm e\phi(0)/4k_BT]}{1 \pm e^{-|z_w|/\lambda_D}\tanh[Z_\pm e\phi(0)/4k_BT]}\right]^2 & \text{Gouy – Chapman} \end{cases} \tag{5.17}$$

For ions bound to extractants at the interface, the ions can be modeled to be located in a mathematical plane $n(z_w) = \sigma_{int}\delta(z_w)$ because this calculation assumes a planner interface. and the top equation in (5.17) reduces to equation (5.9), with x' replaced by x and α' replaced by α. Note that bounded ions can also be modeled to follow Gouy–Chapman distributions with a rather high negative interfacial potential. Figure 37 shows the normalized ion distribution for trivalent cations that have a Debye length of 30 Å, but under different interfacial potentials. For $\phi(0) = -0.05$ V, almost all the ions are within 10 Å from the interface, and the calculated fluorescence intensity from this distribution barely differs from having all the ions bound extractants located at $z = 0$ (red solid and black dashed lines).

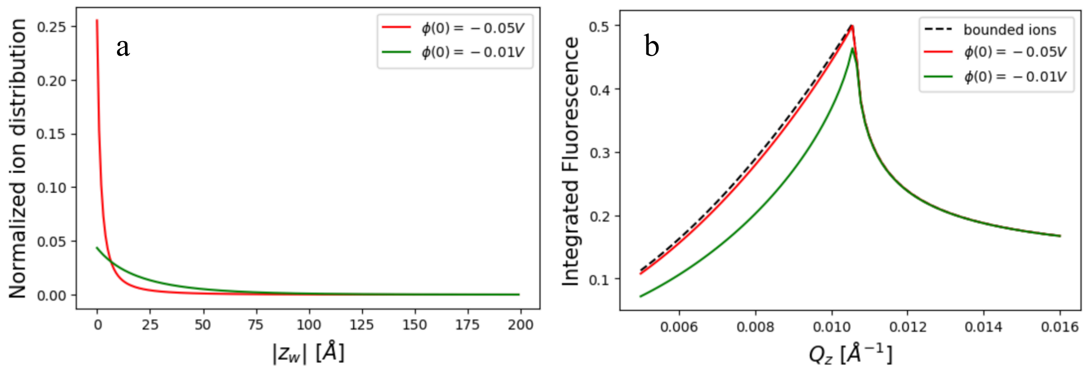


Figure 37. (a) Example of normalized distribution of excess ions with different Debye length; (b) Total fluorescence created by ions in the evanescent region.

Figure 37 (b) shows the dependence of fluorescence intensity on the incident angle. The intensities above the critical angle are the same regardless of the ion distribution. This is because the penetration depth $\Lambda(\alpha)$ becomes $1{\sim}10$ µm (see Figure 5) above critical angle and the exponential decay in equation (5.16) has no effect along z direction. Below the critical angle, the decay of the X-ray intensity the within evanescent region is significant. Ions closer to the interface create more fluorescence intensity. However, there is no significant difference between the slope of the integrated intensity below the critical Q, making it nearly impossible to identify the distribution of the ions in the evanescent region. The intensity profile of the excess ions bounded to the interface can be interpreted as that of the non-bounded excess ions with a higher concentration, and the difference between the two is within measurement error bars.

It is plausible that ion distributions that extend over length scales of 100 to 300 Å, which would be of comparable dimensions to the evanescent decay lengths, would exhibit an integrated fluorescent intensity below the critical Q that has a different shape than shown in Figure 37b. In this case the form of the ion distribution could be probed by XFNTR. Although it is unlikely that these long decay lengths would be present in 1 mM solutions of trivalent electrolytes, they might be expected in dilute solutions of monovalent electrolytes with concentrations of roughly 0.1 mM to 1 mM.

5.5 Conclusion

We have extended the XFNTR method and used it to measure *in situ* the interfacial ion density and the concentrations of ions in neighboring bulk phases. The method measures the fluorescence intensity as a function of incident angle (a Q_z-scan) or the footprint position (equivalently the sample height Δsh) of the incident beam on the interface (an sh-scan). Using this technique, we measured the interfacial ion density and the ion concentrations in the two neighboring bulk phases of a model system for back extraction. Europium ions were detected at the interface for samples with nitric and citric acid solutions, but not for pure water. The Q_z-scan demonstrated that the measured excess ion interfacial density for samples containing citric acid solutions is smaller than for those containing nitric acid solutions. The density of excess interfacial ions in samples with citric acid was close to the statistical limit of the detection capabilities of these Q_z-scan XFNTR measurements. The sh-scan XFNTR measurement was used as a complementary tool to support the existence of excess interfacial ions in the citric acid samples. More sh-scan data near the $\Delta sh = 0$ position is needed in future experiments to fully reveal the interfacial feature. These results suggest that measuring XFNTR data in a two dimensional grid in $(Q_z, \Delta sh)$ space, which could take advantage of the complementary features of both sh-scan and Q_z-scan XFNTR, might reduce the statistical errors on the fitting parameters.

5.6 Cited Literature

1. Todd TA. Development in the U . S . Fuel Cycle Program. 2011.
2. Todd TA. Separations research for advanced nuclear fuel cycles. In: *ACS Symposium Series*. Vol 1046. American Chemical Society; 2010:13-18. doi:10.1021/bk-2010-1046.ch002
3. Mincher BJ, Peterman DR, Mcdowell RG, Olson LG. Radiation Chemistry of Advanced TALSPEAK Flowsheet. 2013:20.
4. Wilson AM, Bailey PJ, Tasker PA, Turkington JR, Grant RA, Love JB. Solvent extraction: The coordination chemistry behind extractive metallurgy. *Chem Soc Rev*. 2014;43(1):123-134. doi:10.1039/c3cs60275c
5. Steytler DC, Jenta TR, Robinson BH, Eastoe J, Heenan RK. Structure of Reversed Micelles

Formed by Metal Salts of Bis(ethylhexyl) Phosphoric Acid. *Langmuir*. 1996;12(6):1483-1489. doi:10.1021/la950669x

6. Gannaz B, Antonio MR, Chiarizia R, Hill C, Cote G. Structural study of trivalent lanthanide and actinide complexes formed upon solvent extraction. *J Chem Soc Dalt Trans*. 2006;(38):4553-4562. doi:10.1039/b609492a

7. Ellis RJ, Anderson TL, Antonio MR, Braatz A, Nilsson M. A SAXS Study of Aggregation in the Synergistic TBP–HDBP Solvent Extraction System. *J Phys Chem B*. 2013;117(19):5916-5924. doi:10.1021/jp401025e

8. Parratt LG. Surface studies of solids by total reflection of x-rays. *Phys Rev*. 1954;95(2):359-369. doi:10.1103/PhysRev.95.359

9. Kiessig H. Interferenz von Röntgenstrahlen an dünnen Schichten. *Ann Phys*. 1931;402(7):769-788. doi:10.1002/andp.19314020702

10. Pershan PS, Schlossman M. *Liquid Surfaces and Interfaces: Synchrotron X-Ray Methods*. Vol 9780521814. Cambridge University Press; 2012. doi:10.1017/CBO9781139045872

11. Buff FP, Lovett RA, Stillinger FH. Interfacial Density Profile for Fluids in the Critical Region. *Phys Rev Lett*. 1965;15(15):621-623. doi:10.1103/physrevlett.15.621

12. Pershan PS, Schlossman M. *Liquid Surfaces and Interfaces*. Cambridge University Press; 2012. doi:10.1017/cbo9781139045872

13. Bu W, Yu H, Luo G, et al. Observation of a rare earth ion-extractant complex arrested at the oil-water interface during solvent extraction. *J Phys Chem B*. 2014;118(36):10662-10674. doi:10.1021/jp505661e

14. https://en.wikipedia.org/wiki/X-ray_fluorescence.

15. https://en.wikipedia.org/wiki/Iohexol.

16. Nilsson M, Nash KL. Review article: A review of the development and operational characteristics of the TALSPEAK process. *Solvent Extr Ion Exch*. 2007;25(6):665-701. doi:10.1080/07366290701634636

17. Philip Horwitz E, Kalina DC, Diamond H, Vandegrift GF, Schulz WW. THE TRUEX PROCESS - A PROCESS FOR THE EXTRACTION OF THE TKANSURANIC ELEMENTS EROM NITRIC AC In WASTES UTILIZING MODIFIED PUREX SOLVENT*. *Solvent Extr Ion Exch*. 1985;3(1-2):75-109. doi:10.1080/07366298508918504

18. Weaver B, Kappelmann FA. TALSPEAK, ORNL-3559. 1964.

19. Gelis A V., Kozak P, Breshears AT, et al. Closing the Nuclear Fuel Cycle with a Simplified Minor Actinide Lanthanide Separation Process (ALSEP) and Additive Manufacturing. *Sci Rep*. 2019;9(1):1-11. doi:10.1038/s41598-019-48619-x

20. Gelis A V., Lumetta GJ. Actinide lanthanide separation process - ALSEP. *Ind Eng Chem Res*. 2014;53(4):1624-1631. doi:10.1021/ie403569e

21. Nash KL. The Chemistry of TALSPEAK: A Review of the Science. *Solvent Extr Ion Exch*. 2015;33(1). doi:10.1080/07366299.2014.985912

22. Ellis RJ. Acid-switched Eu(III) coordination inside reverse aggregates: Insights into a synergistic liquid-liquid extraction system. *Inorganica Chim Acta*. 2017;460:159-164. doi:10.1016/j.ica.2016.08.008

23. Lumetta GJ, Gelis A V., Carter JC, Niver CM, Smoot MR. The Actinide-Lanthanide Separation Concept. *Solvent Extr Ion Exch*. 2014;32(4):333-347. doi:10.1080/07366299.2014.895638

24. Gannaz B, Antonio MR, Chiarizia R, Hill C, Cote G. Structural study of trivalent lanthanide

and actinide complexes formed upon solvent extraction. *J Chem Soc Dalt Trans*. 2006;(38):4553-4562. doi:10.1039/b609492a

25. Antonio MR, McAlister DR, Horwitz EP. An europium(iii) diglycolamide complex: Insights into the coordination chemistry of lanthanides in solvent extraction. *Dalt Trans*. 2015;44(2):515-521. doi:10.1039/c4dt01775g

26. Gullekson BJ, Brown MA, Paulenova A, Gelis A V. Speciation of Select f-Elements with Lipophilic Phosphorus Acids and Diglycol Amides in the ALSEP Backward-Extraction Regime. *Ind Eng Chem Res*. 2017;56(42):12174-12183. doi:10.1021/acs.iecr.7b02379

27. Bu W, Mihaylov M, Amoanu D, et al. X-ray studies of interfacial strontium-extractant complexes in a model solvent extraction system. *J Phys Chem B*. 2014;118(43):12486-12500. doi:10.1021/jp508430e

28. Bu W, Yu H, Luo G, et al. Observation of a rare earth ion-extractant complex arrested at the oil-water interface during solvent extraction. *J Phys Chem B*. 2014;118(36):10662-10674. doi:10.1021/jp505661e

29. Liang Z, Bu W, Schweighofer KJ, et al. Nanoscale view of assisted ion transport across the liquid–liquid interface. *Proc Natl Acad Sci U S A*. 2019;116(37):18227-18232. doi:10.1073/pnas.1701389115

30. Bu W, Hou B, Mihaylov M, et al. X-ray fluorescence from a model liquid/liquid solvent extraction system. *J Appl Phys*. 2011;110(10):1-6. doi:10.1063/1.3661983

31. Bu W, Vaknin D. X-ray fluorescence spectroscopy from ions at charged vapor/water interfaces. *J Appl Phys*. 2009;105(8). doi:10.1063/1.3117487

32. Goebel A, Lunkenheimer K. Interfacial tension of the water/n-alkane interface. *Langmuir*. 1997;13(2):369-372. doi:10.1021/la960800g

33. Zhengshui H, Ying P, Wanwu M, Xun F. Purification of Organophosphorus Acid Extractants. *Solvent Extr Ion Exch*. 1995;13(5):965-976. doi:10.1080/07366299508918312

34. Svantesson I, Persson G, Hagström I, Liljenzin JO. Distribution ratios and empirical equations for the extraction of elements in Purex high level waste solution-II: HDEHP. *J Inorg Nucl Chem*. 1980;42(7):1037-1043. doi:10.1016/0022-1902(80)80397-6

35. Bloch JM, Yun W. Condensation of monovalent and divalent metal ions on a Langmuir monolayer. *Phys Rev A*. 1990;41(2):844-862. doi:10.1103/PhysRevA.41.844

36. Shapovalov VL, Ryskin ME, Konovalov O V., Hermelink A, Brezesinski G. Elemental Analysis within the Electrical Double Layer Using Total Reflection X-ray Fluorescence Technique. *J Phys Chem B*. 2007;111(15):3927-3934. doi:10.1021/jp066894c

37. Bu W, Flores K, Pleasants J, Vaknin D. Preferential Affinity of Calcium Ions to Charged Phosphatidic Acid Surface from a Mixed Calcium/Barium Solution: X-ray Reflectivity and Fluorescence Studies. *Langmuir*. 2009;25(2):1068-1073. doi:10.1021/la803161a

38. Vaknin D, Bu W. Neutrally charged gas/liquid interface by a catanionic langmuir monolayer. *J Phys Chem Lett*. 2010;1(13):1936-1940. doi:10.1021/jz1005434

39. Als-Nielsen J, McMorrow D. *Elements of Modern X-Ray Physics, 2nd Edition*. 2nd ed. Wiley https://www.wiley.com/en-ao/Elements+of+Modern+X+ray+Physics,+2nd+Edition-p-9780470973943. Accessed February 18, 2020.

40. http://henke.lbl.gov/optical_constants/.

41. Chapman D. A contribution to the theory of electrocapillarity. *Philos Mag*. 1913;25:475–481.

42. M G. Sur la constitution de la charge électrique a la surface d'un électrolyte. *J Phys*.

1910;9:457–468.

6 CONCLUSIONS

This thesis describes investigations of metal ion distributions at the liquid-liquid interface in model systems for forward and backward solvent extraction by X-ray surface scattering techniques. For the forward extraction system, we characterized interfacial intermediate states using X-ray reflectivity and fluorescence techniques. We find that trivalent rare earth ions, Y(III) and Er(III), combined with di-hexadecyl phosphoric acid (DHDP) extractants to form inverted bilayer structures at the interface; these appear to be condensed phases of small ion-extractant complexes, an intermediate state in the solvent extraction process. Ions have been transported across the aqueous-organic interface but have not yet been dispersed into the organic phase. In contrast, divalent Sr(II) forms an ion-extractant complex with DHDP that leaves it exposed to the water phase; this result implies that a second process that transports Sr(II) across the interface has yet to be observed. Calculations demonstrate that the budding of reverse micelles formed from interfacial Sr(II) ion-extractant complexes could transport Sr(II) across the interface. Our results suggest a connection between the observed interfacial structures and the extraction mechanism, which ultimately affects the extraction selectivity and kinetics.

We also investigated the interfacial europium ion distribution in a model system for back extraction that mimics chemical conditions within the Actinide-Lanthanide Separation Process (ALSEP). The measurements detected Eu(III) ions at the interfaces with nitric and citric acid solution, though larger density of ions were measured when nitric acid solutions were used. In addition, we report on advances in the X-ray technique used in this study, X-ray Fluorescence Near Total Reflection (XFNTR), which were required to complete the study. These experimental results demonstrate the capability of XFNTR to quantitatively characterize the presence of ions at the liquid-liquid interface in the presence of ions in both adjoining bulk phases, which is a requirement for further *in situ* investigations of the role of the liquid-liquid interface in the ALSEP process.

The study of the back extraction system suggests new directions in the future. First, Q_z-scans in the XFNTR technique can be combined with sh-scans to produce a two-dimensional surface. Fits to this surface are expected to produce results with lower error bars. This may be particularly helpful for measuring the interfacial ion density in samples with citric acid solution which has ions in both bulk phases and whose XFNTR data is relatively featureless. Second, investigations into the effect of different components dissolved in the aqueous solution on the back extraction process can be explored.

Appendix A

A.1 Code for extracting data points from a fluorescence spectrum

This set of code in A.1 is mobile. There are copies of them in every beamtime folder where "mca_profile.py" is customized to specific beamtime.

A.1.1 beamprofile.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


get_ipython().run_line_magic('pylab', 'inline')
from profile import *


# In[7]:


file_name = file_dir + 'Scan 125.txt'
bp = np.loadtxt(file_name)

plt.errorbar(bp[:,0],bp[:,1],bp[:,2],ls='-')
# plt.errorbar(d[:,0],d[:,1],d[:,2],ls='',color='r')
# plt.plot(bp[:,0],init,ls='-',color='r',alpha=0.5)
# plt.plot(bp[:,0],fit,ls='-',color='k')
plt.grid(1)
plt.show()


# In[10]:


cut_range = [[-0.075,-0.055]]
d = ft.fit_range(bp,cut_range)

number_of_peaks = 0
mod,pars = ft.gauss_model(number_of_peaks,d,bg='linear')
pars['li_slope'].set(-5.849,vary=True)
pars['li_intercept'].set(-0.159,vary=True)

init = mod.eval(pars, x=bp[:,0])
out = ft.fit_data(d,mod,pars)
fit = mod.eval(out.params, x=bp[:,0])

plt.errorbar(bp[:,0],bp[:,1],bp[:,2],ls='-')
plt.errorbar(d[:,0],d[:,1],d[:,2],ls='',color='r')
# plt.plot(bp[:,0],init,ls='-',color='r',alpha=0.5)
plt.plot(bp[:,0],fit,ls='-',color='k')
plt.grid(1)
plt.show()
print lm.fit_report(out)


# In[14]:


cut_range = [[-0.10,-0.055]]
d = ft.fit_range(bp,cut_range)

number_of_peaks = 1
mod,pars = ft.gauss_model(number_of_peaks,d,bg='linear')
```

```
pars['li_slope'].set(-4.3246,vary=False)
pars['li_intercept'].set(-0.2364,vary=False)
pars['g1_center'].set(-0.08,min=-0.09,max=-0.07)
pars['g1_sigma'].set(0.01,min=0,max=1)
pars['g1_amplitude'].set(1,min=0,max=10)
init = mod.eval(pars, x=bp[:,0])
out = ft.fit_data(d,mod,pars)
fit = mod.eval(out.params, x=bp[:,0])


# In[19]:


out.params['g1_sigma'].value


# In[15]:


print lm.fit_report(out)


# In[16]:


plt.errorbar(bp[:,0],bp[:,1],bp[:,2],ls='-',alpha=0.5)
plt.errorbar(d[:,0],d[:,1],d[:,2],ls='',color='r')
# plt.plot(bp[:,0],init,ls='-',color='r',lw=2,alpha=0.5)
plt.plot(bp[:,0],fit,ls='-',color='k')
plt.grid(1)
plt.show()


# In[29]:


def trapezoid(x, center, height, top, bottom):
    x = x - center # center x
    a = (x>=-top/2)*(x<=top/2) # mask for top region
    aa = x/np.abs(x) * a # mask for left(-1) and right(+1)
    b = (x>=-bottom/2)*(x<=bottom/2) * aa # mask for shoulder region
    # add the top region and shoulder region
    z = x * a * top + height/(bottom-top)*(bottom-2*(x*b))
    return z

def residual(pars,x,data=None,yerr=None):
    bg_slope = pars['bg_slope']
    bg_const = pars['bg_const']
    center = pars['center'].value
    height = pars['height'].value
    top = pars['top'].value
    bottom = pars['bottom'].value

    model = trapezoid(x,center,height,top,bottom)           + bg_slope * x + bg_const
    if data is None:
        return model
    elif yerr is None:
        return model-data
    else:
        return (model-data)/yerr


# In[30]:


params = lm.Parameters()
params.add_many(('bg_slope', -4.3246, 0, None, None, None, None),
                ('bg_const', -0.2364, 0, None, None, None, None),
                ('center',     -0.09, 1, None, None, None, None),
                ('height',     10.00, 1, None, None, None, None),
                ('top',        0.005, 1, None, None, None, None),
                ('bottom',     0.015, 1, None, None, None, None))
result = lm.minimize(residual, params, args=(bp[:,0],),
                        kws={'data':bp[:,1],'yerr':bp[:,2]})


# In[31]:




# In[ ]:
```

A.1.2 mca_plot-automatic.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


get_ipython().run_line_magic('pylab', 'inline')
# %matplotlib notebook
from mca_profile import *


# In[6]:


data = [] # initialize data list
scans = [312] # add frames in scan into data dict

for scan in scans:
    filename = vortex_dir + mca_head + str(scan) + mca_tail
    q, dd = mca.readMcaScan(filename,calib=calibration)
    for i in range(len(q)):
        data.append((q[i],dd[i]))
qz = np.array([dd[0] for dd in data])

qz


# In[18]:


#!!!!!! plot selected q !!!!!#
qz_indices = [6]
errorbar = True
show_all = 0   # if show all scans
if show_all:
    qz_indices = range(len(qz))


# In[22]:


# plot
fig = plt.figure(figsize=(10,5),dpi=100)
ax = fig.add_subplot(111)
ax.grid(1)
# ax.set_title("fluorescence spectrum for scan %d" %(scans[0]),fontsize=20)
ax.set_xlabel("Energy(KeV)",fontsize=15)
ax.set_ylabel("Fluorescence Intensity",fontsize=15)
ax.set_xlim([4.00,10.000])
ax.set_ylim([0,0.002])
# ax.set_yscale('log')
for i in qz_indices:
    q = qz[i]
    d = data[i][1]
    if errorbar==False: d[2]=0
    if i in [12,13]:
        ax.errorbar(d[0]/1000,d[1],yerr=d[2], markersize=5,
               marker='s', linestyle='', alpha=.5,
               label=str(i)+' qz='+str(qz[i]))
    else:
        ax.errorbar(d[0]/1000.,d[1],yerr=d[2],
                   marker='.', linestyle='', alpha=1,
                   label=str(i)+' qz='+str(qz[i]))
ax.legend(loc="best",fontsize=10)
plt.show()


# In[24]:


#!!!!!! plot out single scan!!!!!#
type_str = 'qz_'
def average(a,b):
    c = np.zeros(a.shape)
    c[0] = a[0]
    c[1] = (a[1] + b[1]) * 1/2
    c[2] = np.sqrt(a[2]**2 + b[2]**2) * 1/2
    return c

def add_integral(x,out):
    global intensity1
    p = out.params
    area1 = peak_integral(p['g1_height'],p['g1_fwhm'],fac=1e5)
    intensity1 = np.vstack((intensity1,np.array([x,area1.n,area1.std_dev])))

intensity1 = np.zeros((1,3)) # container for fluorescence intensity
for idx in qz_indices:
# for idx in [-2]:

    # read data
    cut_range = [[4500,7200],[8500,12000]]
    q, d_fit = fit_range(data,qz,idx,cut_range)
    print(d_fit[-1])
#     cut_plot(qz, idx, errorbar = errorbar,
#              x_range=(3500,12000),
#              y_range=(0,100))

    number_of_models = 4
    mod,pars = build_model(number_of_models)
    try: # if 'out' is defined, use 'out.params'
#         pars = out.params
        pass
    except NameError: # otherwise use 'pars'
        pass
    finally:
        init = mod.eval(pars, x=d_fit[:,0])
        out,comps = fit_data(mod,pars)

#     guess_plot(qz,idx,init,
#                x_range=(3500,9000),
#                y_range=(0,200))

    errorbar_off = False
    fit_plot(qz,idx,errorbar,
             x_range=(3500,9000),
             y_range=(0,200))

    add_integral(q,out)
intensity1 = intensity1[1:]
print('Done!')


# In[23]:


q_str = 'qz{}_'.format(0.015)
type_str = 'sh_'
left = -0.1
right = 0.1
```

```
points = len(intensity1)
sh = np.linspace(left,right,points)
if type_str == 'sh_':
    for i,entry in enumerate(intensity1):
        entry[0] = sh[i]
print(intensity1)
print('\n\n')


# In[24]:


## import numpy as np
import matplotlib.pyplot as plt

#!!! plot (independent code) !!!#
fig = plt.figure(figsize=(10,6),dpi=100,facecolor='white')
ax = fig.add_subplot(111)
ax.set_title('Fluorescence data',fontsize=20)
ax.set_xlabel('Qz(A^-1)',fontsize=15)
ax.set_ylabel('Normalized Intensity',fontsize=15)
ax.grid(1)
# ax.set_xlim([-0.1,0.1])
# ax.set_xlim([0.004,0.017])
# ax.set_ylim([0.00,0.14])
ax.errorbar(intensity1[:,0],intensity1[:,1], yerr=intensity1[:,2],
            color='r',marker='^',linestyle='',
            label='#'+str(scans)[1:-1].replace(', ','&'))

# b = np.loadtxt(work_dir + "sample3_50mMEu(NO3)3_shscan_s1h1_qz0.015_flu.txt")
# ax.errorbar(b[:,0], b[:,1], yerr=b[:,2], color='r',
#             marker='d',linestyle='',label='303&305')
#
# b = np.loadtxt(work_dir + "sample3_50mMEu(NO3)3_shscan_s1h2_qz0.006_289_flu.txt")
# ax.errorbar(b[:,0], b[:,1], yerr=b[:,2], color='g',
#             marker='v',linestyle='',label='#289')
#
# b = np.loadtxt(to_save+".txt")
# ax.errorbar(b[:,0], b[:,1], yerr=b[:,2], color='y',
#             marker='o',linestyle='',label='640')

ax.legend(loc='best')
plt.show()


# In[25]:


sample_str = 'sample{}_'.format('3')

abs_str = 'abs{}_'.format(8)
scan_str = '{}_'.format(scans[0])
sample_description = 'water_calibration'
to_save = type_str +            sample_str +            scan_str +            sample_description +
"s1h0.2_" +           abs_str +           q_str +           "flu"
print(to_save)


# In[52]:


np.savetxt(work_dir + to_save + ".txt", intensity1, fmt="%.4f\t%.4e\t%.4e\t")
fig.savefig(work_dir + to_save + ".png")


# In[ ]:
```

A.1.3 mca_profile.py

```
import sys
import os
import numpy as np
np.set_printoptions(suppress=True)
import matplotlib as mpl
import matplotlib.pyplot as plt
plt.style.use('classic')
mpl.rc('figure',facecolor='white')


import os
import mca_routines as mca
from scipy.optimize import curve_fit
from lmfit import  Model

from mca_plot_routine import *
import fit_routine as ft


# read data into mca_scans with "scan#: mca data(3 rows)" pair
work_dir = '/Users/zhuzi/Documents/work/data/201912Dec/'
vortex_dir = '/Users/zhuzi/Documents/work/data/201912Dec/vortex/'
mca_head = '20191206_'
mca_tail = '_mca'

calibration = [0,8,0] # read from MCA file
```

A.1.4 mca_routines.py

```
#!/usr/bin/env python
'''
This nobg file is used to process ".mca" files.
'''

import sys
import numpy as np
np.set_printoptions(suppress=True)
import subprocess
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

__all__ = ['flatten','chop','readMcaScan','fit_curve']

def expo(x,const,inten,decay,off_set):
    return const + inten*np.exp(-(x-off_set)/decay)


def expDecay(x,const,inten,off_set,decay):
    '''
    y = const + inten * exp(-(x-off_set)/decay)
    '''

    x = np.array(x)
    y = const + inten * np.exp(-(x-off_set)/decay)
    return y

def gaussian(x,position,height,width):
    '''
    y = inten * exp(-(x-off_set)**2/width)
    '''
    c = width / 2.3548 * 1.414
    y = height * np.exp(-(x-position)**2/c**2)
    return y
```

```
def flatten(block,calib=None,raw_count=False):
    '''
    takes in a scan block in an mca file, gives out an two dimensional array
    with rows: channel/counts/errors
    A scan block is a text block of .mca file that starts with "#S".

    calib is used to transform channels into energies, it has to be in the form
    of list, tuple, or an ndarray.
    if calib = None, there is no callibration, data are in channels
    if calib = "in file", use the calibration coefficients in "@CALIB" line.
        E = calib[0] + calib[1] * x + calib[2] * x**2

    Returns
    -------
    flatten: tuple.
             0: qz for this scan block.
             1: a two-dimensonal list with
    '''
    for i,line in enumerate(block):
        if line.startswith('#Q'):
            qz = float(line.split()[3])
        if line.startswith('#Monc'):
            if raw_count: monc = 1 # get the raw counts from detector.
            else: monc = float(line.split()[1]) # monc
        if line.startswith('#@CTIME'):
            real = float(line.split()[1]) # the amount of time it should take
            live = float(line.split()[3]) # the amount of time it actually takes
            try:
                factor = real / live / monc
            except ZeroDivisionError:
               print ("Error: float division by zero, check @CTIME or @Monc")
        if line.startswith('#@CALIB'):
            calib_ = map(float,line.split()[1:])
        # find the data block that starts with "@A"
        if line.startswith('@A '):
            block[i] = line[2:] # adjust the first line of data, take away '@A'.
            start = i  # mark the line where true data begins.
            break
    # calibration
    if calib==None:
        calib = [0,1,0] # linear term only, which is 1.
    elif calib=="in place":
        calib = calib_ # use the calibration factor in file.
    g = lambda x: calib[0] + calib[1] * x + calib[2] * x**2  # calibration, qudratic form

    d = []
    for i,line in enumerate(block[start:]):
        b=[int(i) for i in line.split('\\')[0].split() if len(line)>10]
        d += b # concatenate each data line
    # calculate energy, intensity, err for each channel
    energy = [g(i) for i in range(1,len(d)+1)]
    intensity = [i*factor for i in d]
    err = [np.sqrt(i+i**2/monc)*factor for i in d] # consistant with Wei's code: plmca_15

    return (qz,[energy,intensity,err])

def chop(filename):
    '''
    Takes in an mca filename, returns a list containing blocks of lines of a
    single frame. This returns a block that could be put in "flatten()" to produce
    final data.
    '''
    with open(filename,'r') as fid:
        f = fid.readlines()
    # find the index for each scan line
    scan_index = [i for i,l in enumerate(f) if l.startswith('#S')]

    # divide the whole file into scan blocks.
    scan_blocks = []
    for i,j in enumerate(scan_index):
        if i+1!=len(scan_index):
            scan_blocks.append(f[j:scan_index[i+1]])
        else:
            scan_blocks.append(f[j:])
    return scan_blocks

def readMcaScan(filename,calib=None,raw=False):
    '''
    takes in an mca scan filename, returns the whole data in the form of:
         data[i][energy:intensity:error]
    Also works for those non-scan .mca files( which only have one mca scan)

    Parameters
    ----------
    filename: the file needed to be processed
    calib: calibration used to transform channels into energies, it has to be in the form
            of list, tuple, or an ndarray.
            if calib = None, there is no callibration, data are in channels
            if calib = "in file", use the calibration coefficients in "@CALIB" line.
                E = calib[0] + calib[1] * x + calib[2] * x**2
    raw: if True return the counts as it is in file; if False, normalize counts to monc.

    Returns
    -------
    readMcaScan: tuple.
        0: a list of all the qz's scanned
        1: an ndarray of data corresponding each qz. for each qz, the strucute
            is [energy intensity error]
    '''
    data, q = [],[]
    blocks = chop(filename) # get data blocks in the file.

    for block in blocks:
        qz, d = flatten(block,calib=calib,raw_count=raw) # process each data block
        data.append(d)
        q.append(qz)
    data = np.array(data)
    return (q,data)

def fit_curve(func,x,y,yerr,guess):
    '''
    Fit a data into a function using curve_fit method provided by scipy.optimize.
    Parameters
    ----------
    func: functions to fit into
    x: numpy array. x value
    y: numpy array. y value
    y_err: numpy array. error
    guess: initial guess for the fitting parameters

    Returns
    -------
    fit_curve: tuple
            0: numpy array. parameters that best fit the data
            1: numpy array. error of parameters that best fit the data.
    '''
    pk_par,pk_cov = curve_fit(func,x,y,p0=guess,sigma=yerr)
    pk_err = np.sqrt(np.diag(pk_cov))
    return (pk_par,pk_err)

if __name__ == "__main__":
    filename = '/Users/ZhuZi/work/data/2015_Apr_15ID/vortex/20150408_483_mca'
    q,x = readMcaScan(filename)
    print (np.amax(x[0][2]))
```

A.1.5 mca_plot_routines.py

```
import sys
sys.path.append('/Users/ZhuZi/work/modules/')
import numpy as np
np.set_printoptions(suppress=True)
import matplotlib.pyplot as plt
import os
import mca_routines as mca
from scipy.optimize import curve_fit
import lmfit as lm
from lmfit import  Model,fit_report
from lmfit.models import GaussianModel, ExponentialModel, LinearModel
import copy


def fit_range(data,qz,idx,cut_range):
    '''
    data: input data, all the frames in a mca scan
    qz: indexes of qz
    index: which index to choose
    cut_range: list of ranges in which the data is fitted.
                e.g. [[range1_left,range1_right],[range2_left,range2_right]]
    '''

    global d, q, d_fit
    factor=1e5
    q = qz[idx]
    d = copy.copy(data[idx][1]) # make a copy of selected data for operating later
    d[1] = d[1]*factor # both data and errors are scaled up by factor
    d[2] = d[2]*factor

    #!!! cut off the signal and show !!!#

    d_fit = [0] # basically empty fit range initialized.
    for sub_range in cut_range:
        d_ = np.transpose(d)
        # cut out the signal
        fit_left,fit_right = 0,0
        for i,row in enumerate(d_):
            if row[0]<sub_range[0]:
                fit_left = i
            elif row[0]<sub_range[1]:
                fit_right = i
        try:
            d_fit = np.vstack((d_fit,d_[fit_left:fit_right]))
        except: d_fit = d_[fit_left:fit_right]

    return q, d_fit


def cut_plot(qz, idx, errorbar=True, x_range=(4000,8000),y_range=(0,16)):
    fig = plt.figure(figsize=(16,5),dpi=100)
    ax = fig.add_subplot(111)
    ax.grid(1)
    ax.set_xlim(list(x_range))
    ax.set_ylim(list(y_range))
    ax.set_title("Data & fit")
    ax.set_xlabel('Energy(eV)',fontsize=20)
    ax.set_ylabel('intensity 1e-5',fontsize=20)
    if not errorbar: d[2]=0
    ax.errorbar(d[0],d[1],yerr=d[2],
                marker='o', linestyle='',
                label=str(idx)+' qz='+str(qz[idx]))
    ax.errorbar(d_fit[:,0],d_fit[:,1],yerr=d_fit[:,2],
                marker='o', linestyle='',color='r',
                label='fit range')
    ax.legend(loc="upper left")
    plt.show()

def build_model(model_num,bg='linear'):
################## lmfit model ################
    global mod, init, pars
    while True:
        if bg=='linear':
            bg_mod = LinearModel(prefix='li_')
            pars = bg_mod.guess(d_fit[:,1], x=d_fit[:,0])
            pars.update(bg_mod.make_params())
            pars['li_slope'].set(-1.267e-5,vary=True)
            pars['li_intercept'].set(1.5776,vary=True)
        elif bg=='exponential':
            bg_mod = ExponentialModel(prefix='exp_')
            pars = bg_mod.guess(d_fit[:,1],x=d_fit[:,0])
            pars.update(bg_mod.make_params())
            pars['exp_amplitude'].set(1e-5,vary=True)
            pars['exp_decay'].set(10, vary=True)

        if model_num==0:
            mod = bg_mod; break

        gauss1  = GaussianModel(prefix='g1_')
        pars.update(gauss1.make_params())
        pars['g1_center'].set(5900, min=5500, max=6300)
        pars['g1_sigma'].set(100, min=50,max=1000)
        pars['g1_amplitude'].set(26000, min=4)
        if model_num==1:
            mod = bg_mod + gauss1; break

        gauss2  = GaussianModel(prefix='g2_')
        pars.update(gauss2.make_params())
        pars['g2_center'].set(6470, min=6000, max=7000)
        pars['g2_sigma'].set(120, min=50,max=1000)
        pars['g2_amplitude'].set(20000, min=4)
        if model_num==2:
            mod = bg_mod + gauss1 + gauss2; break

        gauss3  = GaussianModel(prefix='g3_')
        pars.update(gauss3.make_params())
        pars['g3_center'].set(6850, min=6500, max=7200)
        pars['g3_sigma'].set(80, min=50,max=300)
        pars['g3_amplitude'].set(60000, min=4)
        if model_num==3:
            mod = bg_mod + gauss1 + gauss2 + gauss3; break

        gauss4  = GaussianModel(prefix='g4_')
        pars.update(gauss4.make_params())
        pars['g4_center'].set(5200, min=5000, max=5500)
        pars['g4_sigma'].set(100)
        pars['g4_amplitude'].set(60000)
        if model_num==4:
            mod = bg_mod + gauss1 + gauss2 + gauss3 + gauss4; break

    return mod,pars

def guess_plot(qz,idx, init, x_range=(4000,8000),y_range=(0,16)):
    fig = plt.figure(figsize=(16,5),dpi=100)
    ax = fig.add_subplot(111)
    ax.grid(1)
    ax.set_xlim(list(x_range))
    ax.set_ylim(list(y_range))
    ax.set_title("Initial guess")
    ax.set_xlabel('Energy(eV)',fontsize=20)
    ax.set_ylabel('intensity',fontsize=20)
    ax.errorbar(d[0],d[1],yerr=d[2],
               marker='o', linestyle='',
               label=str(idx)+' qz='+str(qz[idx]),zorder=0)
    ax.plot(d_fit[:,0], init,
            marker='', linestyle='-',lw=5, color='y',
            label=str(idx)+' qz='+str(qz[idx]),zorder=10)
    ax.legend(loc="upper left")
    plt.show()

def fit_data(mod, pars):
    global out,comps
    weights = 1/d_fit[:,2] # weight*(y-fit) is minimized in leastsq sense, so weight=1/error
    out = mod.fit(d_fit[:,1],pars, x=d_fit[:,0],weights=None)
    comps = out.eval_components(x=d_fit[:,0])
    return out,comps

def fit_plot(qz,idx,errorbar, x_range=(4000,8000),y_range=(0,16)):
    fig = plt.figure(figsize=(16,10),dpi=100)
    ax1 = fig.add_subplot(211)
    ax1.grid(1)
    ax1.set_xlim(list(x_range))
    ax1.set_ylim(list(y_range))
    # ax.set_yscale("log",nonposy='clip')
    ax1.set_title("Data & fit")
    ax1.set_xlabel('Energy(eV)',fontsize=20)
    ax1.set_ylabel('intensity',fontsize=20)
    if errorbar==False: d[2]=0
    ax1.errorbar(d[0],d[1],yerr=d[2],
              marker='o', linestyle='', alpha=0.5,
              label=str(idx)+' qz='+str(qz[idx]),zorder=0)
    ax1.plot(d_fit[:,0], out.best_fit,
            marker='',linestyle='-',color='r',lw='2',zorder=10)
    ax1.legend(loc="upper left")

    plt.show()

def peak_integral(height,gwhm,fac=1e5):
        from uncertainties import ufloat as uf
        try:
            height = uf(height.value,height.stderr)
        except AttributeError:
            height = uf(height.value,0)
        try:
            width = uf(gwhm.value, gwhm.stderr)
        except AttributeError:
            width = uf(gwhm.value,0)
        area = height*width/2/fac
        return area
```

A.1.6 fit_routine.py

```
import sys
sys.path.append('/Users/ZhuZi/work/modules/')
import numpy as np
np.set_printoptions(suppress=True)
import matplotlib.pyplot as plt
import os
import mca_routines as mca
from scipy.optimize import curve_fit
import lmfit as lm
from lmfit import  Model,fit_report
from lmfit.models import GaussianModel, ExponentialModel, LinearModel
import copy


def fit_range(data,cut_range):
    '''
    data: input data, columns are: [x,y,err]
    index: which index to choose
    cut_range: list of ranges in which the data is fitted.
               e.g. [[range1_left,range1_right],[range2_left,range2_right]]
    '''
    d = data.transpose()
    #!!! cut off the signal and show !!!#

    d_fit = [0] # basically empty fit range initialized.
    for sub_range in cut_range:
        d_ = np.transpose(d)
        # cut out the signal
        fit_left,fit_right = 0,0
        for i,row in enumerate(d_):
            if row[0]<sub_range[0]:
                fit_left = i
            elif row[0]<sub_range[1]:
                fit_right = i
        try:
            d_fit = np.vstack((d_fit,d_[fit_left:fit_right]))
        except: d_fit = d_[fit_left:fit_right]

    return d_fit
```

```
def cut_plot(qz, idx, errorbar=True, x_range=(4000,8000),y_range=(0,16)):
    fig = plt.figure(figsize=(16,5),dpi=100)
    ax = fig.add_subplot(111)
    ax.grid(1)
    ax.set_xlim(list(x_range))
    ax.set_ylim(list(y_range))
    ax.set_title("Data & fit")
    ax.set_xlabel('Energy(eV)',fontsize=20)
    ax.set_ylabel('intensity 1e-5',fontsize=20)
    if not errorbar: d[2]=0
    ax.errorbar(d[0],d[1],yerr=d[2],
                marker='o', linestyle='',
                label=str(idx)+' qz='+str(qz[idx]))
    ax.errorbar(d_fit[:,0],d_fit[:,1],yerr=d_fit[:,2],
                marker='o', linestyle='',color='r',
                label='fit range')
    ax.legend(loc="upper left")
    plt.show()

def gauss_model(model_num,data,bg='linear'):
################### lmfit model #################
    global mod, init, pars
    while True:
        if bg=='linear':
            bg_mod = LinearModel(prefix='li_')
            pars = bg_mod.guess(data[:,1], x=data[:,0])
            pars.update(bg_mod.make_params())
            pars['li_slope'].set(-1.267e-5,vary=True)
            pars['li_intercept'].set(1.5776,vary=True)
        elif bg=='exponential':
            bg_mod = ExponentialModel(prefix='exp_')
            pars = bg_mod.guess(data[:,1],x=data[:,0])
            pars.update(bg_mod.make_params())
            pars['exp_amplitude'].set(1e-5,vary=True)
            pars['exp_decay'].set(10, vary=True)

        if model_num==0:
            mod = bg_mod; break

        gauss1  = GaussianModel(prefix='g1_')
        pars.update(gauss1.make_params())
        pars['g1_center'].set(5900, min=5500, max=6300)
        pars['g1_sigma'].set(100, min=50,max=1000)
        pars['g1_amplitude'].set(26000, min=4)
        if model_num==1:
            mod = bg_mod + gauss1; break

        gauss2  = GaussianModel(prefix='g2_')
        pars.update(gauss2.make_params())
        pars['g2_center'].set(6470, min=6000, max=7000)
        pars['g2_sigma'].set(120, min=50,max=1000)
        pars['g2_amplitude'].set(20000, min=4)
        if model_num==2:
            mod = bg_mod + gauss1 + gauss2; break

        gauss3  = GaussianModel(prefix='g3_')
        pars.update(gauss3.make_params())
        pars['g3_center'].set(6850, min=6500, max=7200)
        pars['g3_sigma'].set(80, min=50,max=300)
        pars['g3_amplitude'].set(6000, min=4)
        if model_num==3:
            mod = bg_mod + gauss1 + gauss2 + gauss3; break

        gauss4  = GaussianModel(prefix='g4_')
        pars.update(gauss4.make_params())
        pars['g4_center'].set(5200, min=5000, max=5500)
        pars['g4_sigma'].set(100)
        pars['g4_amplitude'].set(500)
        if model_num==4:
            mod = bg_mod + gauss1 + gauss2 + gauss3 + gauss4; break

    return mod,pars
```

```
def fit_data(data,mod,pars):

    weight = 1/data[:,2] # weight*(y-fit) is minimized in leastsq sense, so weight=1/error
    out = mod.fit(data[:,1],pars, x=data[:,0],weights=weight)
    #components = out.eval_components(x=data[:,0])
    return out #,components

def fit_plot(qz,idx,errorbar, x_range=(4000,8000),y_range=(0,16)):
    fig = plt.figure(figsize=(16,10),dpi=100)
    ax1 = fig.add_subplot(211)
    ax1.grid(1)
    ax1.set_xlim(list(x_range))
    ax1.set_ylim(list(y_range))
    # ax.set_yscale("log",nonposy='clip')
    ax1.set_title("Data & fit")
    ax1.set_xlabel('Energy(eV)',fontsize=20)
    ax1.set_ylabel('intensity',fontsize=20)
    if errorbar==False: d[2]=0
    ax1.errorbar(d[0],d[1],yerr=d[2],
                 marker='o', linestyle='', alpha=0.5,
                 label=str(idx)+' qz='+str(qz[idx]),zorder=0)
    ax1.plot(d_fit[:,0], out.best_fit,
             marker='',linestyle='-',color='r',lw='2',zorder=10)
    ax1.legend(loc="upper left")

    plt.show()
def peak_integral(height,gwhm,fac=1e5):
        from uncertainties import ufloat as uf
        height = uf(height.value,height.stderr)
        width = uf(gwhm.value,gwhm.stderr)
        area = height*width/2/fac
        return area
```

A.2 Code for XFNTR software

A.2.1 main.py

```
import sys
import os

#######################################################
# define an exception hook to prevent the app from crashing on exception
#  reference:  https://stackoverflow.com/questions/38020020/pyqt5-app-exits-on-error-where-pyqt4-
app-would-not
sys._excepthook = sys.excepthook
def exception_hook(exctype, value, traceback):
    print(exctype, value, traceback)
    sys._excepthook(exctype, value, traceback)
sys.excepthook = exception_hook

# Use absolute path instead of relative path ('./') to avoid trouble when installed by pip
dir_path = os.path.dirname(os.path.realpath(__file__))
sys.path.append(dir_path)

from PyQt5.QtCore import pyqtRemoveInputHook
from PyQt5.QtWidgets import QApplication
from mainwindow import MainWindow

def main():
    # create application
    pyqtRemoveInputHook()
    app = QApplication(sys.argv)
    app.setApplicationName('15ID-C XFNTR Analyzer')

    # create widget
    w = MainWindow()
    w.setWindowTitle('15ID-C XFNTR Analyzer')
    # w.setWindowIcon(QIcon('logo.png'))
    w.show()

    # connection
    app.lastWindowClosed.connect(app.quit)

    # execute application
    sys.exit(app.exec_())

if __name__ == '__main__':

    main()
```

A.2.2 mplwidget.py

```
# from PyQt4 import QtGui

from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

import matplotlib
matplotlib.use("Qt4Agg")
from matplotlib.backends.backend_qt5agg \
 import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
from matplotlib.backends.backend_qt4agg import NavigationToolbar2QT as NavigationToolbar

class MplCanvas(FigureCanvas):
	def __init__(self):
		self.fig = Figure()
		self.ax = self.fig.add_subplot(111)
		FigureCanvas.__init__(self, self.fig)
		FigureCanvas.setSizePolicy(self, QSizePolicy.Expanding, QSizePolicy.Expanding)
		FigureCanvas.updateGeometry(self)

class MplWidget(QWidget):
	def __init__(self, parent = None):
		QWidget.__init__(self, parent)
		self.canvas = MplCanvas()
		self.toolbar = NavigationToolbar(self.canvas, parent)
		self.vbl = QVBoxLayout()
		self.vbl.addWidget(self.canvas)
		self.vbl.addWidget(self.toolbar)
		self.setLayout(self.vbl)
```

A.2.3 fit_ref.py

```
import sys
import math as m
import numpy as np
from numba import jit
from numba.typed import List
import matplotlib.pyplot as plt
import periodictable
from lmfit import minimize, Parameters, Parameter, report_fit, fit_report
from scipy.special import erf

e_radius = periodictable.constants.electron_radius*1e10
energy = 20 # keV


#### data generating ####

def sldCalFun(d,rho,sigma,x):

    N = len(rho) - 1 # number of internal interfaces
    z = [0] + [sum(d[:j+1]) for j in range(len(d))] # height of each interface
    sld_tot = np.zeros(x.shape)

    # sld calculation model in Wei's paper
    for i in range(N):
        erfx = (x-z[i])/sigma[i]/np.sqrt(2)
        sld = 0.5 * (erf(erfx) * (rho[i+1]-rho[i]))
        sld_tot = sld_tot + sld
    sld_tot = sld_tot + 0.5 * (rho[0]+rho[N])

    return sld_tot

def refCalFun(d,rho,mu,sigma,x,rrf=False):

    if sigma[0] <= 0: sigma[0] = 1e-5 # eliminate zero
    sigma = sigma[0] * np.ones(len(sigma)) # fixed sigma

    erad = e_radius
    slab=0.25

    x_ = List()
    for xx in x: x_.append(xx)

    k0=2*np.pi*float(energy)/12.3984 # wave vector
    lamda=2*np.pi/k0
    theta=x/2/k0   # convert q to theta

    # total length of inner slabs plus 4 times roughness for both sides
    length = np.sum(d) + 4* (sigma[0]+sigma[-1])
    steps=int(length/slab) # each sliced box has thickness of ~ 0.25 A
    xsld=np.linspace(-4*sigma[0],np.sum(d)+4*sigma[-1],steps) # get the z-axis for sld

    sd=length/steps # thickness for each slab
    intrho=sldCalFun(d,rho,sigma,xsld) # rho for all the steps
    intmu=sldCalFun(d,mu,sigma,xsld) # mu for all the steps


    # was....
    d, sdel, sbet = List(), List(), List()
    sdel.append(float(rho[0])) # delta for the top phase
    sbet.append(float(mu[0]/2/k0/1e8))        # beta for the top phase
    for rho_ in intrho: # add rho for the interface
        sdel.append(float(rho_))
    sdel.append(float(rho[-1]))  # delta for the bottom phase
    # add delta for the interface
    for mu_ in intmu: # add beta for the interface
        sbet.append(float(mu_/2/k0/1e8))
    sbet.append(float(mu[-1]/2/k0/1e8))    # beta for the bottom phase
    for i in sdel: d.append(slab)
    ref,refr=parratt(x_, lamda, d, sdel, sbet)

    d_, sdel_, sbet_ = List(), List(), List()
    for i in [0.0,1.0]: d_.append(i)
    for i in [sdel[0],sdel[-1]]: sdel_.append(i)
    for i in [sbet[0],sbet[-1]]: sbet_.append(i)
    frsnll,frsnl1=parratt(x_, lamda,d_, sdel_, sbet_)

    if rrf == True:
        return ref/frsnll
    else:
        return ref

def ref2min(params,x,y,yerr,fit=True):

    ndata = len(x)

    # allocate parameters to different lists
    sigma_t, rho_b, mu = [], [], []
    d, sigma, rho, qoff = [], [], [], []

    par = params.valuesdict()
    for p in par:
        if p.startswith('sigma'):
            if p.endswith('0'):
                sigma_t.append(par[p]); continue
            else:
                sigma.append(par[p]); continue
        if p.startswith('d'):
            d.append(par[p]); continue
        if p.startswith('mu'):
            mu.append(par[p]); continue
        if p.startswith('qoff'):
            qoff.append(par[p]); continue
        if p.startswith('rho'):
            if '_b' in p:
                rho_b.append(par[p]); continue
            else:
                rho.append(par[p]); continue

    if fit==True: # return residual: 1D array
        residual = np.array([])
        for i in range(ndata):
            yy = refCalFun(d,
                           rho+[rho_b[i]],
                           mu,
                           [sigma[i]]+sigma_t,
                           x[i])
            residual = np.append(residual, (yy-y[i])/yerr[i])
        return residual
    else: # return model: (y1,y2,y3,y4,y5)
        model = []
        for i in range(ndata):
            yy = refCalFun(d,
                           rho + [rho_b[i]],
                           mu,
                           [sigma[i]] + sigma_t,
                           x[i])
            model.append(yy)
        return tuple(model)

def iterCallBack(params,iteration,resid,x,y,err,fit=True):
    if fit== False: return None
    m = sum([params[p].vary for p in params]) # number of parameters
    n = sum([len(xx) for xx in x]) # number of points
    df = n - m # degree of freedom
    redchisq = sum(resid**2) / df
    if (iteration<=10) or (iteration%10==0):  #display reduced chisq every 10 iteration
        print(iteration,redchisq)

def initParameters(name_list,para_list):

    if len(name_list)!=len(para_list):
        print(" Does name_list and value_list match, please check ")
        return
    params = Parameters()
    for i,name in enumerate(name_list):
        p = para_list[i]
        params.add(name,value=p[0],vary=p[1],min=p[2],max=p[3])
    return params

def updateParameters(mrefpar,refparaname,refpara):
    '''Return the parameter name for the items selected in the table.'''
    ui = mrefpar  # it only accepts ref_multiFit_par.ui as ui

    parameter_list = refpara
    name_list = refparaname
    index_dict = name2index(name_list,reverse=True)

    # update the value of each cell in the table
    for index in index_dict:
        row, col = index
        value = float(ui.parTW.item(row,col).text())
        i = name_list.index(index_dict[index])
        parameter_list[i][0] = value # update parameter value
        parameter_list[i][1] = False # clear the vary status first

    # update "vary" status in the selected cells
    selected_items = ui.parTW.selectionModel().selectedIndexes()
    selected_names = []
    for item in selected_items:
        selected_index = (item.row(), item.column())
        selected_name = index_dict[selected_index]
        i = name_list.index(selected_name)
        parameter_list[i][1] = True # update vary status

    return parameter_list

def name2index(name_list, reverse=False):
    ''' Create a mapping from table index to parameter name, or vice versa.
    If reverse=False, create a dict in the form of {name:position}, otherwise
    {position:name}.'''
    ndata = len([p for p in name_list if p.startswith('qoff')])
    nlayer = len([p for p in name_list if p.startswith('d')])

    index_dict = {}
    k = 0
    for row in range(ndata+nlayer+1):
        for col in range(4):
            if (row==0) & (col==0): continue
            elif (row>nlayer) & ((col==0)|(col==3)): continue
            else:
                index_dict[name_list[k]] = (row,col)
                k = k + 1
    if reverse == False:
        return index_dict
    else:
        name_dict = {v:k for k,v in index_dict.iteritems()}
        return name_dict

def readData(rrf_files,sel_rows,fit_range,err_type=0):

    '''read multiple data set and cut them to fit range'''

    if sel_rows == []: return None
    if fit_range == None: fit_range = [0,1]
    # import pdb; pdb.set_trace();
    rrf = [np.loadtxt(rrf_files[r],comments='#') for r in sel_rows]
    for i,d in enumerate(rrf):
        select = (d[:,0]>=fit_range[0])&(d[:,0]<=fit_range[1])
        rrf[i] = d[select]
    qz = tuple([a[:,0] for a in rrf]) # tuple: (qz1,qz2,...)
    y = tuple([a[:,1] for a in rrf]) # tuple: (data1,data2,...)
    if err_type==0: # tuple: (err1,err2,...)
        yerr = tuple([a[:,2] for a in rrf])
    elif err_type==1:
        yerr = tuple([np.sqrt(a[:,1]) for a in rrf])
    elif err_type==2:
        yerr = tuple([a[:,1] for a in rrf])
    else:
        yerr=tuple([np.ones(a[:,0].shape) for a in rrf])

    return (qz, y, yerr)

@jit(nopython=True)
def parratt(q, lamda, d, rho, beta):
    """
        Calculation of reflectivity by Parrat Recursion Formula without any roughness.
        Directly translated from a fortran subroutine 'parratt' in xr_ref.f90, which is
    attached as following:

    subroutine parratt(q,lambda,d,rho,beta,Rgen,Rgenr,M,N)
    !***************************************************************************
    !Calculation of reflectivity by Parratt Recursion Formula without any roughness
    !
    !M = No. of data points
    !N = No. of slabs
    !lambda = wavelength
    !d = list of thicknesses of each slab
    !rho=list of Electron densities of each slab
    !beta=list of Absorption coefficient in each slab
    !Rgen = generated reflectivtiy data
    !Rgenr= generated reflectance data
    !q = change in wave vector
    !***************************************************************************
    integer :: M,N
    double precision :: q(0:M), Rgen(0:M)
    double precision :: d(0:N+1), rho(0:N+1), beta(0:N+1), qc2(0:N+1)
    double precision :: lambda
    double complex :: X, fact1, fact2, r(0:N+1), k1, k2, fact,Rgenr(0:M)
    double precision, parameter :: re=2.817938e-5, pi=3.14159

    do j=0,N+1
       qc2(j)=16.0d0*pi*re*(rho(j)-rho(0))
    enddo

    do i = 0,M
       r(N+1)=dcmplx(0.0d0,0.0d0)
       do j=N,0,-1
          k1=cdsqrt(dcmplx(q(i)**2-qc2(j),-32.0d0*beta(j)*pi**2/lambda**2))
          k2=cdsqrt(dcmplx(q(i)**2-qc2(j+1),-32.0d0*beta(j+1)*pi**2/lambda**2))
          X=(k1-k2)/(k1+k2)
          fact1=dcmplx(dcos(dble(k2)*d(j+1)),dsin(dble(k2)*d(j+1)))
          fact2=dexp(-aimag(k2)*d(j+1))
          fact=fact1*fact2
          r(j)=(X+r(j+1)*fact)/(1.0+X*r(j+1)*fact)
       enddo
       Rgenr(i)=r(0)
       Rgen(i)=cdabs(r(0))**2
    enddo
    end subroutine parratt

    """
    M = len(q) - 1
    N = len(d) - 2
    r = np.ones(N+2) * (0+0j) # this definition is compatible to numba
    qc2 = np.zeros(N+2)
    Rgen = np.zeros(M+1)
    Rgenr = np.ones(M+1) * (0+0j)

    for j in range(N+2):
        qc2[j] = 16.0 * np.pi * e_radius * (rho[j] - rho[0])


    for i in range(M+1):
        r[N+1] = 0+0j
        for j in range(N,-1,-1):
           k1 = np.sqrt(q[i]**2-qc2[j] - 32.0*beta[j]*np.pi**2/lamda**2*1j)
           k2 = np.sqrt(q[i]**2-qc2[j+1] - 32.0*beta[j+1]*np.pi**2/lamda**2*1j)
           X = (k1 - k2) / (k1 + k2)
           fac1 = m.cos(k2.real*d[j+1]) + m.sin(k2.real*d[j+1])*1j
           fac2 = m.exp(-k2.imag*d[j+1])
           fact = fac1 * fac2
           r[j] = (X + r[j+1]*fact) / (1.0+X*r[j+1]*fact)
        Rgenr[i] = r[0]
        Rgen[i] = abs(r[0])**2
    return (Rgen, Rgenr)




################################################################################
################################################################################

if __name__ == '__main__':
    rho_t = 0.25913738441154344
    rho_b = 0.337
    itMu = 2.792661024598891e-09
    ibMu = 7.0179999999999995e-09
    qz = np.array([0.00879463, 0.00878271, 0.00877078, 0.00875884, 0.00874688,
       0.0087349 , 0.00872291, 0.0087109 , 0.00869887, 0.00868683,
       0.00867477, 0.00866269, 0.0086506 , 0.00863849, 0.00862636,
       0.00861422, 0.00860205, 0.00858987, 0.00857768, 0.00856546,
       0.00855323, 0.00854098, 0.00852872, 0.00851643, 0.00850413,
       0.00849181, 0.00847948, 0.00846712, 0.00845475, 0.00844236,
       0.00842995, 0.00841752, 0.00840507, 0.00839261, 0.00838012,
        0.00836762, 0.0083551 , 0.00834256, 0.00833   , 0.00831742,
        0.00830483, 0.00829221, 0.00827958, 0.00826692, 0.00825425,
        0.00824156, 0.00822884, 0.00821611, 0.00820336, 0.00819059,
        0.0081778 , 0.00816498, 0.00815215, 0.0081393 , 0.00812643,
        0.00811353, 0.00810062, 0.00808769, 0.00807473, 0.00806176,
        0.00804876, 0.00803574, 0.0080227 , 0.00800964, 0.00799656,
        0.00798346, 0.00797033, 0.00795719, 0.00794402, 0.00793083,
        0.00791762, 0.00790439, 0.00789113, 0.00787785, 0.00786455,
        0.00785123, 0.00783788, 0.00782452, 0.00781112, 0.00779771,
        0.00778427, 0.00777081, 0.00775733, 0.00774382, 0.00773029,
        0.00771673, 0.00770316, 0.00768955, 0.00767593, 0.00766227,
        0.0076486 , 0.0076349 , 0.00762118, 0.00760743, 0.00759365,
        0.00757985, 0.00756603, 0.00755218, 0.0075383 , 0.0075244 ,
        0.00751048, 0.00749652, 0.00748255, 0.00746854, 0.00745451,
        0.00744045, 0.00742637, 0.00741226, 0.00739812, 0.00738396,
        0.00736977, 0.00735555, 0.0073413 , 0.00732703, 0.00731272,
        0.00729839, 0.00728403, 0.00726965, 0.00725523, 0.00724079,
        0.00722631, 0.00721181, 0.00719728, 0.00718272, 0.00716813,
        0.00715351, 0.00713886, 0.00712418, 0.00710947, 0.00709473])
    ref = refCalFun([], [rho_t, rho_b], [itMu,ibMu], [3.0], qz)
print(ref)
```

A.2.4 flu_geometry_routines.py

```
import numpy as np
import matplotlib.pyplot as plt

def laser_gun(origin, alpha, wall=None,celling=None):
    '''
        Define a laser starting from a point with a certain angle,
    until it hits a wall/celling.
        Input: origin: np.array([x,y])
        Returns the x/y of the hitting point.
    '''
    if wall != None:
        x = wall
        y = origin[1] + np.tan(alpha) * (x - origin[0])
    elif celling != None:
        y = celling
        x = origin[0] + 1/np.tan(alpha) * (celling - origin[1])
    else:
        print("please define a stop")
        return None
    return np.array([x, y])


def central_dist(point, line):
    '''
    Calculate the distance from a point to a line.
    Line is defined as: A*x + B*y + C = 0.
    Input line takes in (A,B,C), point takes in [x,y]
    '''
    x, y = point[0], point[1]
    A, B, C = line
    d = np.abs(A*x+B*y+C)/np.sqrt(A**2+B**2)
    return d


def surf_points(x,R):
    '''return an array of surface points given x coordinates
       Input: x, 1D array; curvature in m '''
    x = np.array(x)
    theta = -x / R # sin(theta) = theta, no difference
    y = - R * theta**2/2  # cos(theta) = 1 - theta**2/2, no difference
    surface = np.array([x,y])
    return np.transpose(surface), theta

def hit_surface(point, alpha, curvature, span, sh=0):
    '''
        Given a point and angle, calculate its laser interactig with the surface
        input: points, 1-D array
        output: points, 2-D array
        source: http://mathworld.wolfram.com/Circle-LineIntersection.html.
    Note that this algorithm applies to circle center at (0,0), sh is the sample height relative
NOM position.

    '''
    # if point[0]==0.: return np.array([0,0])
    R = curvature
    cell_left = - span / 2
    cell_right = span / 2
    pt1 = np.array([point[0],point[1]+(R-sh)]) # raise the line up by R.
    pt2 = laser_gun(pt1,alpha,wall=-50) # second point for the line

    dx = pt2[0] - pt1[0]
    dy = pt2[1] - pt1[1]
    dr = np.sqrt(dx**2+dy**2)
    D = pt1[0] * pt2[1] - pt2[0] * pt1[1]
    delta = (R*dr)**2-D**2
    if delta < 0:
        hit = np.array([np.inf,np.inf])
    else:
        x1 = (D*dy + dy/abs(dy)*dx*np.sqrt(delta)) / dr**2
        y1 = (-D*dx + abs(dy)*np.sqrt(delta)) / dr**2
#     x2 = (D*dy + sign(dy)*dx*np.sqrt(delta)) / dr**2
#     y2 = (D*dx + abs(dy)*np.sqrt(delta)) / dr**2
        hit = np.array([x1,y1-(R-sh)]) # put the line down by R-sh.
        if hit[0] > cell_right or hit[0] < cell_left:
            hit = np.array([hit[0], np.inf])
return hit
```

A.2.5 flu_routines_new.py

```
from collections import OrderedDict
import periodictable as pdtb
import lmfit as lm
import multiprocessing

r_e = pdtb.constants.electron_radius * 1e10  # classical electron radius, in A
N_A = pdtb.constants.avogadro_number  # Avogadro number, unitless
k_B = 1.38065e-23  # Boltzman constant, in J/K
p_igMu = (1/1.717)*1e-7    # absorption coefficient of 20keV beam by glass
p_thick = 3.5 * 1e7     # thickness of glass tray in A

# define an function that will be used in the program a lot.
absorb = lambda x: np.nan_to_num(np.exp(x))

import numpy as np
import scipy.stats as stat
import fit_ref as mfit
import flu_geometry_routines as gm



def penetrate(beta, delta, alpha, k0):
    alpha[alpha == np.inf] = 0
    alpha = alpha.astype(complex)
    beta_top, beta_bot = beta
    delta_top, delta_bot = delta
    alpha_c = np.sqrt(2 * (delta_bot - delta_top))
    trans = 4 * np.abs(alpha / (alpha + np.sqrt(alpha ** 2 - alpha_c ** 2))) ** 2
    penetration_coeff = 2 * k0 * np.imag(np.sqrt(alpha ** 2 - alpha_c ** 2 + beta_bot * 2j))
    return 1/penetration_coeff, trans

def update_flu_parameters(p, *args):

    assert type(p) is OrderedDict
    # *args have to be at least fitting parameters
    flu_par = args[0]

    # update the fitting parameter whatsoever
    try:
        p['hisc'] = flu_par['hisc'].value  # scale factor for the top phase, unitless.
        p['losc'] = flu_par['losc'].value  # scale factor for the bottom phase, unitless.
        p['bg'] = flu_par['bg'].value  # background intensity, unitless.
        p['tC'] = flu_par['upbk'].value  # ion concentration of the top phase, in M.
        p['bC'] = flu_par['lobk'].value  # ion concentration of the bottom phase, in M.
        p['sC'] = flu_par['surd'].value  # ion surface number density, in A^-2.
        p['qoff'] = flu_par['qoff'].value  # q off set for the data
        p['soff'] = flu_par['soff'].value * 1e7  # det range offset for the measurement
        p['l2off'] = flu_par['loff'].value * 1e7  # l2 offset for the measurement
        p['curv'] = flu_par['curv'].value * 1e10  # the curvature of the interface, in A.
        if p['curv'] == 0: p['curv'] = 10000 * 1e10
    except KeyError as e:
        print("Please check your parameter:{}".format(e))

    if len(args) == 1:
        return p

    # if the *args is tuple (flu_par, sys_par, flu_elements), do the following
    try:
        sys_par = args[1]
        flu_elements = args[2]
    except IndexError:
        print("update_flu_parameters takes 3 extra arguments!")

    # parameterize beam profile
    width = sys_par['width'] * 1e7  # width or FWHM of the beam, in A
    beam_profile = sys_par['beam']
    steps = 500
    if beam_profile == 'Uniform':
        beam_size = width
        weights = np.ones(steps + 1)
    elif beam_profile == 'Gaussian':
        stdev = width / 2.355  # FWHM of the beam, i.e. 2.355 sigma
        beam_size = 2 * (3 * stdev)  # keep the beam up to +/-3 standard deviation, or 99.73% of
intensity.
        rays = np.linspace(-beam_size/2, beam_size/2, steps+1)  # devide the beam into 500 rays
        weights = stat.norm(0, stdev).pdf(rays) * width  # weight normalized to the total intensity
    else:
        print("Error: please choose the right beam profile: uniform/gaussian")
        return None

    # unwrap system parameters
    E_inc = float(sys_par['E_inc'])  # energy of incidence, in KeV
    E_emit = float(sys_par['E_emt'])  # energy of  emission, in KeV
    mu_top_inc = float(sys_par['mu_top_inc'] / 1e8)  # abs. coef. of top phase for incidence, in
1/A
    mu_top_emit = float(sys_par['mu_top_emt'] / 1e8)  # abs. coef. of top phase for emission, in
1/A
    mu_bot_inc = float(sys_par['mu_bot_inc'] / 1e8)  # abs. coef. of bot phase for incidence, in
1/A
    mu_bot_emit = float(sys_par['mu_bot_emt'] / 1e8)  # abs. coef. of bot phase for emission, in
1/A
    rho_top = sys_par['rho_top']  # electron density of top pahse, in A^-3
    rho_bot = sys_par['rho_bot']  # electron density of top pahse, in A^-3

    det_len = sys_par['det_len'] * 1e7  # detector length, in A
    span = sys_par['span'] * 1e7

    # calculate abosrption parameters.
    k0 = 2 * np.pi * E_inc / 12.3984  # wave vector for incidence, in A^-1
    k1 = 2 * np.pi * E_emit / 12.3984  # wave vector for emission, in A^-1

    # construct elemental parameters
    vol_top, vol_bot = 0, 0  # vol_bot for ions for 1 L subphase
    ne_top, ne_bot = 0, 0  # total number of electrons from 1 L subphase
    bet_top_inc_ele, bet_top_emit_ele = 0, 0  # beta: for top phase at inc.& emit.
    bet_bot_inc_ele, bet_bot_emit_ele = 0, 0  # beta: for bot phase at inc.& emit.
    flupara_ele = {}  # setup dict for all elements in the subphase
    for i, e in enumerate(flu_elements):
        flupara_ele[i] = \
            [e[0], float(e[1]), float(e[2]),  # name, composition, ionic radius
             pdtb.elements.symbol(e[0]).number,
             pdtb.elements.symbol(e[0]).xray.scattering_factors(energy=sys_par['E_inc'])[1],
            pdtb.elements.symbol(e[0]).xray.scattering_factors(energy=sys_par['E_emt'])[1]]
    for i, p_ in flupara_ele.items():
        n_top_density = p['tC'] * p_[1] * N_A / 1e27  # atoms per A^3 in top phase
        n_bot_density = p['bC'] * p_[1] * N_A / 1e27  # atoms per A^3 in bot phase
        vol_top += n_top_density * 4 / 3 * np.pi * p_[2] ** 3
        vol_bot += n_bot_density * 4 / 3 * np.pi * p_[2] ** 3
        ne_top += n_top_density * p_[3]  # electrons per A^3
        ne_bot += n_bot_density * p_[3]  # electrons per A^3
        bet_top_inc_ele += n_top_density * 2 * np.pi * r_e * p_[4] / k0 ** 2
        bet_top_emit_ele += n_top_density * 2 * np.pi * r_e * p_[5] / k1 ** 2
        bet_bot_inc_ele += n_bot_density * 2 * np.pi * r_e * p_[4] / k0 ** 2
        bet_bot_emit_ele += n_bot_density * 2 * np.pi * r_e * p_[5] / k1 ** 2
    # absorption coefficient and electron density modified by solvent.
    rho_top = ne_top + (1 - vol_top) * rho_top
    rho_bot = ne_bot + (1 - vol_bot) * rho_bot

    # re-evaluate mu
    mu_top_inc = 2 * k0 * bet_top_inc_ele + (1 - vol_top) * mu_top_inc
    mu_top_emit = 2 * k1 * bet_top_emit_ele + (1 - vol_top) * mu_top_emit
    mu_bot_inc = 2 * k0 * bet_bot_inc_ele + (1 - vol_bot) * mu_bot_inc
    mu_bot_emit = 2 * k1 * bet_bot_emit_ele + (1 - vol_bot) * mu_bot_emit

    # calculate beta
    bet_top_inc = mu_top_inc / k0 / 2
    bet_top_emit = mu_top_emit / k1 / 2
    bet_bot_inc = mu_bot_inc / k0 / 2
    bet_bot_emit = mu_bot_emit / k1 / 2

    # calculate delta
    del_top_inc  =  2  *  np.pi  *  r_e  *  rho_top  /  k0  **  2   #  del=2*PI*re*rho/k^2,  unitless
#self.flutopdel
    del_top_emit = 2 * np.pi * r_e * rho_top / k1 ** 2
    del_bot_inc = 2 * np.pi * r_e * rho_bot / k0 ** 2
    del_bot_emit = 2 * np.pi * r_e * rho_bot / k1 ** 2

    p['k0'] = k0 # incident ray Energy, in KeV.
    p['k1'] = k1 # emission ray energy, in KeV.
    p['detR'] = det_len # detector range, in mm.
    p['wt'] = weights  # weights for the beam profile, the length of which is the total amount of
steps.
    p['bmsz'] = beam_size  # the size of the beam for footprint calculation, in A.
    p['tRho'] = rho_top # electron density of top phase, in A^-3.
    p['bRho'] = rho_bot # electron density of bottom phase, in A^-3.
    p['itMu'] = mu_top_inc # mu for incident beam in top phase, in cm^-1
    p['etMu'] = mu_top_emit # mu for emitted beam in top phase, in cm^-1
    p['ibMu'] = mu_bot_inc # mu for incident beam in bottom phase, in cm^-1
    p['ebMu'] = mu_bot_emit # mu for emitted beam in bottom phase, in cm^-1
    p['itBt'] = bet_top_inc # beta for incident beam in top phase, in cm^-1
    p['etBt'] = bet_top_emit # beta for emitted beam in top phase, in cm^-1
    p['ibBt'] = bet_bot_inc # beta for incident beam in bottom phase, in cm^-1
    p['ebBt'] = bet_bot_emit # beta for emitted beam in bottom phase, in cm^-1
    p['itDt'] = del_top_inc # delta for incident beam in top phase, in cm^-1
    p['etDt'] = del_top_emit # delta for emitted beam in top phase, in cm^-1
    p['ibDt'] = del_bot_inc # delta for incident beam in bottom phase, in cm^-1
    p['ebDt'] = del_bot_emit # delta for emitted beam in bottom phase, in cm^-1
    p['span'] = span # the length of the sample cell, "the span", in A.

    return p

def fluCalFun_core(a0,sh,p):

    '''takes in flupara_fit, qz, return fluorescence data.
       Note that 'weights' contains the info of the steps for integration
       a0: the incident angle of X-ray beam, corrected with Qz_offset, in rad.
       sh: the sample height shift w.r.t. its norminal position, in A.
       p['wt']: weights for the beam profile, the length of which is the total amount of steps.
       p['k0']: wavevector for incident ray Energy, in KeV.
       p['detR']: detector range, in mm.
       p['hisc']: scale factor for the top phase, unitless.
       p['losc']: scale factor for the bottom pahse, unitless.
       p['bg']: background intensity, unitless.
       p['k1']: wavevector for emission ray energy, in KeV.
       p['tC']: ion concentration of the top phase, in M.
```

```
        p['bC']: ion concentration of the bottom phase, in M.
        p['sC']: ioin surface number density, in A^-2.
        p['qoff']: q off set for the data
        p['soff']: sample height offset (effoct of detector offset included) for the measurement
        p['l2off']: l2 offset for the measurement
        p['tRho']: electron density of top phase, in A^-3.
        p['bRho']: electron density of bottom phase, in A^-3.
        p['itMu']: mu for incident beam in top pahse, in A^-1
        p['etMu']: mu for emitted beam in top phass, in A^-1
        p['ibMu']: mu for incident beam in bottom pahse, in A^-1
        p['ebMu']: mu for emitted beam in bottom phass, in A^-1
        p['itBt']: beta for incident beam in top pahse, in A^-1
        p['etBt']: beta for emitted beam in top phass, in A^-1
        p['ibBt']: beta for incident beam in bottom pahse, in A^-1
        p['ebBt']: beta for emitted beam in bottom phass, in A^-1
        p['itDt']: delta for incident beam in top pahse, in A^-1
        p['etDt']: delta for emitted beam in top phass, in A^-1
        p['ibDt']: delta for incident beam in bottom pahse, in cm^-1
        p['ebDt']: delta for emitted beam in bottom phass, in cm^-1
        p['span']: the length of the sample cell, "the span", in A.
        p['curv']: the curvature of the interface, in A.
        p['bmsz']: the size of the beam for footprint calculation, in A.
        '''


    steps = len(p['wt']) - 1
    fprint = p['bmsz'] / np.sin(a0) # foortprint of the beam on the interface.
    stepsize = fprint / steps
    center = - sh / a0
    del_detR = 0 # horizontal shift of detector.

    # initialize fluorescence data, rows: total, aqueous, organic, interface


    # get the position of single ray hitting the surface
    x0 = np.linspace(center-fprint/2, center+fprint/2, steps+1) # beam is equivalently shited by 
"center"
    surface = np.array([gm.hit_surface([xx, 0], -a0, p['curv'], p['span']) for xx in x0])
    hit = np.isfinite(surface[:,0]) * np.isfinite(surface[:,1])  # rays that hit the liquid-liquid 
interface
    # block = np.isfinite(surface[:,0]) * np.isinf(surface[:,1])  # rays that are blocked by the 
tray
    block = (surface[:,0]<-p['span']/2) * np.isinf(surface[:,1])  # rays that are blocked by the 
tray
    x_s = surface[:, 0][hit]  # x' for rays that hit on the interface
    z_s = surface[:, 1][hit] # z' for rays that hit on the interface
    wt_s = p['wt'][hit]  # weight for rays that hit on the interface
    x_g = surface[:, 0][block] # x' for rays blocked by glass tray
    z_g = surface[:, 1][block] # z' for rays blocked by glass tray
    wt_g = p['wt'][block] # weight for rays that are blocked by the glass tray

    # (x,z) and other surface geometry for points where beam hit at the interface.
    theta = -x_s / p['curv']  # incident angle
    a_new = a0 + theta  # actual incident angle w.r. to the surface
    # a1 = a0 + 2 * theta
    a1 = a0
    x_inc = x_s + z_s / a0  # x position where the inc. xray passes z=0 line
    x_ref = x_s - z_s / a1  # x position where the ref. xray passes z=0 line.
    x_inc_g = x_g + z_g / a0  # x position where the inc. xray passes z=0 line
    x_ref_g = x_g - z_g / a1  # x position where the ref. xray passes z=0 line.


    mu_eff_inc = p['etMu'] + p['itMu'] / a0  # eff.abs.depth for incident beam in oil phase
    mu_eff_ref = p['etMu'] - p['itMu'] / a1  # eff.abs.depth for reflected beam in water phase

    # z coordinate of the intersection of ray with following:
    # detector range shifted by 'del_detR' is the same as beam shifted by '-del_detR'.
    z_inc_l = (x_inc - del_detR + p['detR']/2) * a0  # incidence with left det. boundary: x=-l/2
    z_inc_r = (x_inc - del_detR - p['detR']/2) * a0  # incidence with right det. boundary: x=l/2
    z_ref_l = (-x_ref + del_detR - p['detR']/2) * a1  # reflection with left det. boundary: x=-l/2
    z_ref_r = (-x_ref + del_detR + p['detR']/2) * a1  # reflection with right det. boundary: x=l/2
    z_inc_l_g = (x_inc_g - del_detR + p['detR']/2) * a0  # incidence with left det. boundary: x=-
l/2
```

```
    z_inc_r_g = (x_inc_g - del_detR- p['detR']/2) * a0  # incidence with right det. boundary: x=l/2
    z_ref_l_g = (-x_ref_g + del_detR - p['detR']/2) * a1  # reflection with left det. boundary:
x=-l/2
    z_ref_r_g = (-x_ref_g + del_detR + p['detR']/2) * a1  # reflection with right det. boundary:
x=l/2

    # two regions: region3: [-h/2a0,-l/2] & region 2: [-l/2,l/2]
    x_region = [(x_s <= (-p['detR']/2+del_detR)),\
                (x_s > (-p['detR']/2+del_detR)) * (x_s < (p['detR']/2+del_detR))]

    ################### for redgion 1: region x>= l/2  #######################
    x0_region1 = x0[surface[:, 0] > p['detR']/2]  # choose x0 with x'>l/2
    wt_region1 = p['wt'][surface[:, 0] > p['detR']/2]  # choose weight with x'>l/2
    upper_bulk1 = wt_region1 * \
                 absorb(-x0_region1 * p['itMu']) / mu_eff_inc * \
                 (absorb((x0_region1 + p['detR']/2) * a0 * mu_eff_inc) -
                  absorb((x0_region1 - p['detR']/2) * a0 * mu_eff_inc))

    # define the initial intensity to be just background value
    flu = np.array([p['bg']] * 7)

    # if beam miss the surface entirely, do the following:
    if len(x_s) == 0:  # the entire beam miss the interface, only incidence in upper phase.
        # sh_offset_factor = absorb(-mu_top_emit * center[i] * a0)
        usum_inc = stepsize * np.sum(upper_bulk1)
        flu[3] += p['hisc'] * usum_inc * N_A * p['tC'] / 1e27   # oil phase incidence only +
background
        flu[0] = flu[3]  # total intensity only contains oil phase
        return flu

    ref = mfit.refCalFun([], [p['tRho'], p['bRho']], [p['itMu'], p['ibMu']], [3.0], 2 * p['k0'] *
a_new)
    p_depth, trans = penetrate((p['itBt'],p['ibBt']), (p['itDt'],p['ibDt']), a_new, p['k0'])
    p_depth_eff = 1 / (p['ebMu'] + a_new/a0 / p_depth)

    ################### for region -l/2 < x < l/2  #################
    lower_bulk2 = x_region[1] * wt_s * absorb(-x_s * p['itMu'] - z_s / p_depth) * trans * p_depth
* \
                 (absorb(z_s / p_depth_eff) - absorb(z_inc_r / p_depth_eff))
    interface = x_region[1] * wt_s * trans * absorb(-p['itMu'] * x_s)
    upper_bulk2_inc = x_region[1] * wt_s * \
                     (absorb(-x_inc * p['itMu']) / mu_eff_inc * (
                             absorb(z_inc_l * mu_eff_inc) - absorb(z_s * mu_eff_inc)))
    upper_bulk2_inc[np.isnan(upper_bulk2_inc)] = 0  # if there is nan, set to 0
    upper_bulk2_ref = x_region[1] * wt_s * \
                     (absorb(-x_ref * p['itMu']) / mu_eff_ref * ref * (
                             absorb(z_ref_r * mu_eff_ref) - absorb(z_s * mu_eff_ref)))
    upper_bulk2_ref[np.isnan(upper_bulk2_ref)] = 0  # if there is nan, set to 0

    ###################### for region x<=-l/2 #######################
    lower_bulk3 = x_region[0] * wt_s * absorb(-x_s * p['itMu'] - z_s / p_depth) * trans * p_depth_eff
* \
                 (absorb(z_inc_l / p_depth_eff) - absorb(z_inc_r / p_depth_eff))
    upper_bulk3 = x_region[0] * wt_s * absorb(-x_ref * p['itMu']) / mu_eff_ref * ref * \
                 (absorb(mu_eff_ref * z_ref_r) - absorb(mu_eff_ref * z_ref_l))


    # if there are rays that are blocked by tray, their intensity is still significent
    if np.sum(block) > 0:
        edge = None
    # combine the two regions and integrate along x direction by performing np.sum.
    bsum = stepsize * np.sum(lower_bulk3 + lower_bulk2)
    ssum = stepsize * np.sum(interface)
    usum_inc = stepsize * (np.sum(upper_bulk1) + np.sum(upper_bulk2_inc))
    usum_ref = stepsize * (np.sum(upper_bulk3) + np.sum(upper_bulk2_ref))

    # add the rays blocked by glass tray
    thick_glass = 3.5 * 10e7 # thickness of the side of glass tray, in A.
    mu_ig = 5.824 * 10e-8 # absorption coefficient of glass at 20keV, in A^-1.
    if len(wt_g) != 0:
        mu_eff_w = p['ebMu'] + p['ibMu'] / a0  # eff.abs.depth for incident beam in oil phase
        lower_bulk_glass = 1/mu_eff_w * absorb(p['itMu']*(p['span']/2+thick_glass)) \
                         * absorb(-mu_ig*thick_glass) \
                    * absorb(-p['ibMu']*(p['span']/2+z_g/a0+x_g)) \
                    * (absorb(mu_eff_w*z_inc_l_g) - absorb(mu_eff_w*z_inc_r_g))
        gsum = stepsize * np.sum(lower_bulk_glass)
        int_lobk_gls = p['losc'] * gsum * N_A * p['bC'] / 1e27 # blocked by glasses
    else:
        int_lobk_gls = 0

    # vectorized integration method is proved to reduce the computation time by a factor of 5 to
10.
    int_lobk = p['losc'] * bsum * N_A * p['bC'] / 1e27 # bulk water phase
    int_upbk_inc = p['hisc'] * usum_inc * N_A * p['tC'] / 1e27  # metal ions in the upper phase.
    int_upbk_ref = p['hisc'] * usum_ref * N_A * p['tC'] / 1e27  # metal ions in the upper phase.
    int_sur = p['losc'] * ssum * p['sC']

    flu += np.array([int_lobk + int_lobk_gls + int_sur + int_upbk_inc + int_upbk_ref,
                     int_lobk,  # 3. lower bulk
                     int_sur,  # 4. interface
                     int_upbk_inc+int_upbk_ref, # 5. upper bulk
                     int_upbk_inc,  # 6. upper bulk incidence
                     int_upbk_ref,  # 7. upper bulk reflection
                     int_lobk_gls])
    return flu

def flu2min(pars, x, p, data=None, eps=None): # residuel for flu fitting
    sh, qz = x
    p = update_flu_parameters(p, (pars))

    alpha = (qz + p['qoff']) / p['k0'] / 2 # include the qz offset.
    a0006 = 0.006 / p['k0'] / 2  # incident angle for qz=0.006

    # initialize fluorescence data 3-D matrix
    flu = np.zeros((len(sh), len(qz), 9))
    try:
        for i, ds in enumerate(sh):
            for j, a0 in enumerate(alpha):
                dsh = -p['l2off'] * (a0 - a0006) + p['soff'] + ds*1e7
                flu[i, j, 0] = ds
                flu[i, j, 1] = qz[j]
                flu[i, j, 2:] = fluCalFun_core(a0, dsh, p)
    except KeyError as e:
        print("Please check parameter: {}".format(e))
    if data is None:
        return flu
    if eps is None:
        return (flu[:,:,2].flatten() - data)

    return (flu[:,:,2].flatten() - data) / eps

def fluErrorFitSingleCore2(i, value_list, sh, qz, pname, flu_par, flucal_par, data_to_fit):
    flu_par[pname].value = value_list[i]  # change value of the chosen parameter
    fluerr_result = lm.minimize(fl.flu2min, flu_par,
                                args=((sh, qz), flucal_par),
                                kws={'data': data_to_fit[:, 1], 'eps': data_to_fit[:, 2]})
    return ([i, value_list[i], fluerr_result.nfree, fluerr_result.redchi])

def multiCore(func, iterable):
    pool = multiprocessing.Pool()
    result = pool.map(func, range(len(iterable)))
    pool.close()
    pool.join()
    return result

if __name__ == '__main__':
    '''
    This piece of code does the job of testing. Feel free to change it for different sample
    setup.
    '''
    import os
    import matplotlib.pyplot as plt
    from mpl_toolkits import mplot3d

    p = OrderedDict()
    sys_par = OrderedDict(
                [('E_inc',      20.0),
                 ('E_emt',      5.842),
                 ('mu_top_inc', 0.273),
                 ('mu_top_emt', 7.451),
                 ('mu_bot_inc', 0.7018),
                 ('mu_bot_emt', 26.58),
                 ('rho_top',    0.2591),
                 ('rho_bot',    0.348),
                 ('width',      0.01),
                 ('det_len',    12.7),
                 ('beam',       'Gaussian'),
                 ('span',       75.6)]
              )
    flu_par = lm.Parameters()
    # add with tuples: (NAME VALUE VARY MIN  MAX  EXPR  BRUTE_STEP)
    flu_par.add_many(
                ('losc', 5.27e-9 ,  False, None, None, None, None),
                ('hisc', 5.27e-9,  False, None, None, None, None),
                ('lobk', 0.0545,    False, None, None, None, None),
                ('upbk', 0.0545,   False, None, None, None, None),
                ('surd', 0.0,   False, None, None, None, None),
                ('bg',   0.0,  False, None, None, None, None),
                ('qoff', 3.37e-4,   False, None, None, None, None),
                ('curv', 0,     False, None, None, None, None),
                ('loff', 5.0e-3,    False, None, None, None, None),
                ('soff', 0,    False, None, None, None, None)
            )
    flu_elements = [['Eu', 1, 0.947]]
    p = update_flu_parameters(p, flu_par, sys_par, flu_elements)


    type_ = 'sh'

    dir_path = os.path.dirname(os.path.realpath(__file__)) # the current directory
    data_file = 'q_sample03_312_50mMEu(NO3)3_s1h0.2_flu.txt'
    data = np.loadtxt(os.path.join(dir_path,'test',data_file))

    qz = np.linspace(0.005, 0.016, 10)
    sh = np.linspace(-0.05, 0.05, 10)
    flu = flu2min(flu_par, (sh,qz), p)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.plot_surface(flu[:,:,0],flu[:,:,1],flu[:,:,2],
                    rstride=1, cstride=1,
                    cmap='viridis',edgecolor='none')
    ax.set_title('surface')
    ax.set_xlabel('sh')
    ax.set_ylabel('qz')
    ax.set_zlabel('fluorescence')

    plt.show()


#    fig = plt.figure()
#    ax = fig.add_subplot(111)
##   ax.errorbar(data[:,0],data[:,1],yerr=data[:,2],ls='',marker='.')
#    if type_ == 'sh':
#        qz = np.array([0.006])
##       sh = np.linspace(-0.1, 0.1, 100)
#        sh = np.array([0.0])
#        flu =flu2min(flu_par, (sh,qz), p)
#        ax.set_xlim([-0.11,0.11])
#        ax.plot(flu[:,0,0],flu[:,0,2],ls='-',marker='.')
#        ax.plot(flu[:,0,0],flu[:,0,8],ls='-',marker='.')
#    elif type_ == 'qz':
#        qz = np.linspace(0.005, 0.016, 100)
#        sh = np.array([flu_par['soff']])
#        flu =flu2min(flu_par, (sh,qz), p)
#        ax.set_xlim([0.004,0.018])
#        ax.plot(flu[0,:,1],flu[0,:,2],ls='-')
#        ax.plot(flu[0,:,1],flu[0,:,8],ls='-')
#    ax.grid()
#    plt.show()
```

A.2.6 mainwindow.py

```
import sys
import os
# Use absolute path instead of relative path ('./') to avoid trouble when installed by pip
dir_path = os.path.dirname(os.path.realpath(__file__)) # the current directory
dir_path_test = os.path.join(dir_path,'test')
print(dir_path)
UI_path = dir_path + '/GUI/'
import time
import multiprocessing


######################################################
# This block of code is needed for properly working with PyInstaller

# import the following three modules in order to work with PyInstaller

# mplwidget is imported explicitly here because PyInstaller needs to find it.

# Define function to import external files when using PyInstaller.
#  https://stackoverflow.com/questions/37888581/pyinstaller-ui-files-filenotfounderror-errno-2-no-
such-file-or-directory
def resource_path(relative_path):
    """ Get absolute path to resource, works for dev and for PyInstaller """
    try:
        # PyInstaller creates a temp folder and stores path in _MEIPASS
        base_path = sys._MEIPASS
    except Exception:
        # when not bundled by PyInstaller, normal method is used.
        base_path = os.path.abspath(".")
    return os.path.join(base_path, relative_path)


#######################################################
from collections import OrderedDict

from PyQt5 import uic
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

from scipy.interpolate import interp1d
from scipy.special import *
import numpy as np
import matplotlib.pyplot as plt
from cycler import cycler
default_cycler = (cycler(color=['b','g','c','m','y','k']))
plt.rc('axes', prop_cycle=default_cycler)
plt.rc('lines',markersize=3, linewidth=1)

import lmfit as lm # fitting module
import periodictable as pdtb # a module for periodic table
r_e = pdtb.constants.electron_radius * 1e10  # classical electron radius, in A
N_A = pdtb.constants.avogadro_number  # Avogadro number, unitless
k_B = 1.38065e-23  # Boltzman constant, in J/K

# user defined module
import flu_routines_new as fl

# Here the absolute path is used because PyInstaller needs to find it.
(Ui_MainWindow, QMainWindow) = uic.loadUiType(resource_path(UI_path + 'mainwindow.ui'))


class myThread(QThread):
    def __init__(self,func):
        QThread.__init__(self)
        self.func = func
    def run(self):
        self.func

def fluErrorFitSingleCore2(i, value_list, sh, qz, pname, flu_par, flucal_par, data_to_fit):
        flu_par[pname].value = value_list[i]  # change value of the chosen parameter
        fluerr_result = lm.minimize(fl.flu2min, flu_par,
                                    args=((sh, qz), flucal_par),
                                    kws={'data': data_to_fit[:, 1], 'eps': data_to_fit[:, 2]})
        return ([i, value_list[i], fluerr_result.nfree, fluerr_result.redchi])

def multiCore(func, iterable):
    pool = multiprocessing.Pool()
    result = pool.map(func, range(len(iterable)))
    pool.close()
    pool.join()
    return result


class MainWindow (QMainWindow):
    """MainWindow inherits QMainWindow"""

    def __init__(self, parent = None):
        QMainWindow.__init__(self, parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.tabWidget.setCurrentIndex(1)
        self.directory=os.getcwd()

        self.halftab = '    '
        self.flusavefitindex = 0
        self.flufiles = []
        self.flufitfiles = []
        self.fludata = []
        self.flufitdata = []
        self.beam = 'uniform'
        self.flucal_par = OrderedDict()
        self.flu = 0
        self.selectedflufiles_rows = []
        self.selectedflufitfiles_rows = []
        self.eleradius = pdtb.constants.electron_radius*1e10
        self.avoganum = pdtb.constants.avogadro_number
        self.boltzmann = 1.38065e-23
        self.errorlist = np.array([[1, 1.074], [2, 1.204], [3, 1.222],
                                   [4, 1.220], [5, 1.213], [6, 1.205],
                                   [7, 1.198], [8, 1.191], [9, 1.184],
                                   [10, 1.178], [11, 1.173], [12, 1.168],
                                   [13, 1.163], [14, 1.159], [15, 1.155],
                                   [16, 1.151], [17, 1.148], [18, 1.145],
                                   [19, 1.142], [20, 1.139], [22, 1.134],
                                   [24, 1.129], [26, 1.125], [28, 1.121],
                                   [30, 1.118], [32, 1.115], [34, 1.112],
                                   [36, 1.109], [38, 1.106], [40, 1.104],
                                   [42, 1.102], [44, 1.100], [46, 1.098],
                                   [48, 1.096], [50, 1.094], [60, 1.087],
                                   [70, 1.081], [80, 1.076], [90, 1.072],
                                   [100, 1.069], [120, 1.063], [140, 1.059],
                                   [160, 1.055], [180, 1.052]]) #, [3000, 1.050]])

        self.setupUI()
        self.updatePar()
        self.debugErr()

    def setupUI(self):
        self.ui.addflufilePB.clicked.connect(self.addFluFile)
        self.ui.flufileLW.itemSelectionChanged.connect(self.updateSelectedFluFile)
        self.ui.rmflufilePB.clicked.connect(self.removeFluFile)
        self.ui.addflufitfilePB.clicked.connect(self.addFluFitFile)
        self.ui.flufitfileLW.itemSelectionChanged.connect(self.updateSelectedFluFitFile)
        self.ui.rmflufitfilePB.clicked.connect(self.removeFluFitFile)
        self.ui.fluxaxisCB.currentIndexChanged.connect(self.updateUI)
        self.ui.fluqcCB.stateChanged.connect(self.updateFluPlot)
        self.ui.flulineCB.stateChanged.connect(self.updateFluPlot)
        self.ui.flulegendCB.stateChanged.connect(self.updateFluPlot)
        self.ui.flulegendlocCoB.currentIndexChanged.connect(self.updateFluPlot)
        self.ui.flulogyCB.stateChanged.connect(self.updateFluPlot)
        self.ui.flugridCB.stateChanged.connect(self.updateFluPlot)
        self.ui.flushowCB.stateChanged.connect(self.updateFluPlot)
        self.ui.flucompCB.stateChanged.connect(self.updateFluPlot)
        self.ui.flusimuPB.clicked.connect(self.updateFluCal)
        self.ui.flufitPB.clicked.connect(self.fitFlu)
        self.ui.flusaveCB.activated.connect(self.saveFlu)
        self.ui.fluloadCB.activated.connect(self.loadFlu)
        self.ui.insflusubPB.clicked.connect(self.insFluIon)
        self.ui.rmflusubPB.clicked.connect(self.rmFluIon)
        self.ui.fluErrPB.clicked.connect(self.fluErrorInit)
        self.ui.flulimitPB.clicked.connect(self.setupLimitsUI)
        # connect system parameter signals
        self.ui_syspar = OrderedDict(
            [('E_inc',self.ui.fluIncEnLE),
             ('E_emt',self.ui.fluEmtEnLE),
             ('mu_top_inc',self.ui.flumutopincLE),
             ('mu_top_emt', self.ui.flumutopemtLE),
             ('mu_bot_inc', self.ui.flumubotincLE),
             ('mu_bot_emt',self.ui.flumubotemtLE),
             ('rho_top', self.ui.flurhotopLE),
             ('rho_bot', self.ui.flurhobotLE),
             ('width', self.ui.fluwidthLE),
             ('det_len',self.ui.fludetLE)]
        )
        for p,u in self.ui_syspar.items():
            u.returnPressed.connect(self.updatePar)
            u.returnPressed.connect(self.updateFluCal)
        self.ui.flubmpfCombo.currentIndexChanged.connect(self.updatePar)

        # connect fitting parameter signals
        self.ui_params = OrderedDict(
            [('losc', [self.ui.flubotscaleLE, self.ui.flubotscaleCB]),
             ('hisc', [self.ui.flutopscaleLE, self.ui.flutopscaleCB]),
             ('lobk', [self.ui.flubotbulkLE, self.ui.flubotbulkCB]),
             ('upbk', [self.ui.flutopbulkLE, self.ui.flutopbulkCB]),
             ('surd', [self.ui.flusurdLE, self.ui.flusurdCB]),
             ('bg',   [self.ui.flubgLE, self.ui.flubgCB]),
             ('qoff', [self.ui.fluqoffLE, self.ui.fluqoffCB]),
             ('curv', [self.ui.flucurvLE, self.ui.flucurvCB]),
             ('loff', [self.ui.fluloffLE, self.ui.fluloffCB]),
             ('soff', [self.ui.flusoffLE, self.ui.flusoffCB])]
        )
        for p,u in self.ui_params.items():
            u[0].returnPressed.connect(self.updatePar)
            u[0].returnPressed.connect(self.updateFluCal)
            u[1].stateChanged.connect(self.updatePar)

        # set up a container for parameter limits
        self.par_limits = OrderedDict()
        for p,u in self.ui_params.items():
            if p in ['lobk','upbk','surd','bg','curv']:
                self.par_limits[p] = [0, 1000]
            elif p in ['hisc','losc']:
                self.par_limits[p] = [1e-10, 1000]
            else:
                self.par_limits[p] = [None, None]

    def updateUI(self,index=None, fresh=True):
        '''
        This function also works as a slot for qcombobox signal, so the first arg/kws by default
        is reserved for receiving signal argument, which can be either index or text.
        '''
        print("{}, {}".format(index,fresh))
        # set text for fitting parameters
        for p, u in self.ui_params.items():
            if p in ['curv', 'loff']:
                u[0].setText(format(self.flu_par[p].value, '.4f'))
            else:
                u[0].setText(format(self.flu_par[p].value, '.2e'))

        # set text for system parameters
        for p, u in self.ui_syspar.items():
            u.setText(format(self.sys_par[p], '.4f'))

        # set beam profile
        index = self.ui.flubmpfCombo.findText(self.beam)
        self.ui.flubmpfCombo.setCurrentIndex(index)
        self.xaxis = self.ui.fluxaxisCB.currentText()

        if fresh == True:
            if self.xaxis == 'Qz':
                _xrange = '0.005:0.016'
                self.ui.fluloffCB.setCheckable(True)
                self.ui.fluloffCB.setText('L2 offset')
                self.ui.fluloffqzlabel.setText('mm')
            elif self.xaxis == 'Sh':
                _xrange = '-0.1:0.1'
                self.ui.fluloffCB.setText('Qz')
                self.ui.fluloffLE.setText(format(0.006, '.4f'))
                self.ui.fluloffqzlabel.setText(self.ui.fluqofflabel.text())
            self.ui.fluSimuRangeLE.setText(_xrange)
            self.ui.fluFitRangeLE.setText(_xrange)
        else:
            try:
                _simu_range = ':'.join(str(self.flu_simu_range)[1:-1].split(','))
                _fit_range = ':'.join(str(self.flu_fit_range)[1:-1].split(','))
                self.ui.fluSimuRangeLE.setText(_simu_range)
                self.ui.fluFitRangeLE.setText(_fit_range)
#               self.ui.fluloffLE.setText(str(self.qz[0]))
            except Exception as e:
                self.updateUI(fresh=True)

        self.updatePar()

    def updatePar(self):  #initialize the flu parameters

        # system parameters for fluorescence
        self.beam = str(self.ui.flubmpfCombo.currentText()) # beam profile
        self.sys_par = OrderedDict()
        for p, u in self.ui_syspar.items():
            self.sys_par[p] = float(u.text())
        self.sys_par['beam'] = self.ui.flubmpfCombo.currentText()
        self.sys_par['span'] = 75.6 # the length of sample cell, in mm.

        # fitting parameters for fluorescence
        self.flu_par = lm.Parameters()
        try:
            for name,par in self.ui_params.items():
                self.flu_par.add(name, value=float(par[0].text()), vary=par[1].isChecked(),
                                       min=self.par_limits[name][0],
max=self.par_limits[name][1],
                                       expr=None, brute_step=None)
        except ValueError as VE:
            print("ValueError: ", VE)
        else:
            # fitting (if any) parameters for reflectivity
            self.ref_par = lm.Parameters()
                # add tuples:       (NAME       VALUE   VARY MIN  MAX  EXPR BRUTE_STEP)
            self.ref_par.add_many( ('rho_t', self.sys_par['rho_top'], 0, None, None, None, None),
                                   ('rho_b', self.sys_par['rho_bot'], 0, None, None, None, None),
                                   ('mu_t',  self.sys_par['mu_top_inc'],  0,  None,  None,  None,
None),
                                   ('mu_b',  self.sys_par['mu_bot_inc'],  0,  None,  None,  None,
None),
                                   ('sigma0', 3.0, 0, None, None, None, None),
                                   ('q_off', 0, 0, None, None, None, None ))
            # info of element in the system
            self.flu_elements = [['Eu', 1, 0.947]]  # name, composition, Ionic Radius(A)

        # update the list of sh and qz
        try:
            self.flu_simu_range          =          [float(i)          for          i          in
str(self.ui.fluSimuRangeLE.text()).split(':')]
            self.flu_fit_range = [float(i) for i in str(self.ui.fluFitRangeLE.text()).split(':')]
        except:
            print("Error: Check if the range is right.")
        self.xaxis = self.ui.fluxaxisCB.currentText()
        if self.xaxis == 'Qz':
            self.sh = np.array([0])
            self.qz = np.linspace(self.flu_simu_range[0], self.flu_simu_range[1], 200)
        elif self.xaxis == 'Sh':
            self.qz = np.array([float(self.ui.fluloffLE.text())])
            self.sh = np.linspace(self.flu_simu_range[0], self.flu_simu_range[1], 200)

        # update parameters for flurescence calculation
        self.flucal_par = fl.update_flu_parameters(self.flucal_par,
                                                   self.flu_par,
                                                   self.sys_par,
                                                   self.flu_elements)

    def updateLimits(self):
        ui = self.ui_limits
        try:
            for p, u in self.ui_par_limits.items():
                if u[0].isChecked():
                    try:
                        self.par_limits[p][0] = float(u[1].text())
                        self.par_limits[p][1] = float(u[2].text())
                        assert self.par_limits[p][1] >= self.par_limits[p][0]
                    except ValueError as e:
                        print("{} Please provide a valid limit for '{}'. ".format(e,u[0].text()))
                        raise
                    except AssertionError:
                        print("Max should be larger than Min for {}".format(u[0].text()))
                        raise
                else:
                    self.par_limits[p][0] = None
                    self.par_limits[p][1] = None
        except:
            print("An Error occurs... See above")
        else:
            self.updatePar()
            ui.close()

    def setupLimitsUI(self):

        ui = uic.loadUi(UI_path + 'err4.ui', QDialog(self))
        ui.cancelPB.clicked.connect(ui.close)
        ui.confirmPB.clicked.connect(self.updateLimits)
        self.ui_par_limits = OrderedDict(
            [('losc', [ui.flubotscaleCB, ui.minbotscaleLE, ui.maxbotscaleLE]),
             ('hisc', [ui.flutopscaleCB, ui.mintopscaleLE, ui.maxtopscaleLE]),
             ('lobk', [ui.flubotbulkCB, ui.minbotbulkLE, ui.maxbotbulkLE]),
             ('upbk', [ui.flutopbulkCB, ui.mintopbulkLE, ui.maxtopbulkLE]),
             ('surd', [ui.flusurdCB, ui.minsurdLE, ui.maxsurdLE]),
             ('bg', [ui.flubgCB, ui.minbgLE, ui.maxbgLE]),
             ('qoff', [ui.fluqoffCB, ui.minqoffLE, ui.maxqoffLE]),
             ('curv', [ui.flucurvCB, ui.mincurvLE, ui.maxcurvLE]),
             ('loff', [ui.fluloffCB, ui.minloffLE, ui.maxloffLE]),
             ('soff', [ui.flusoffCB, ui.minsoffLE, ui.maxsoffLE])]
        )
        for p, u in self.ui_par_limits.items():
            if self.par_limits[p][0] is None and self.par_limits[p][1] is None:
                u[0].setChecked(False)
                continue
            else:
                u[0].setChecked(True)
            try:
                u[1].setText(format(self.par_limits[p][0], '.2e'))
            except TypeError:
                pass  # leave empty if limit is None
            try:
                u[2].setText(format(self.par_limits[p][1], '.2e'))
            except TypeError:
                pass
        self.ui_limits = ui
        ui.show()
    def addFluFile(self): #add flu files into the listwidget and deselect all flu files in the
listwidget

        f, _ = QFileDialog.getOpenFileNames(
            caption='Select Multiple Fluorescence Files to import',
            directory=self.directory,
            filter='Flu Files (*.flu*;*_flu.txt)'
        )
        self.flufiles = self.flufiles + f
        self.directory = str(QFileInfo(self.flufiles[0]).absolutePath())
        self.updateFluFile()

    def updateFluFile(self): # update flu files in the listwidget
        self.ui.flufileLW.clear()
        for i, f in enumerate(self.flufiles):
            try:
                self.ui.flufileLW.addItem('#'+str(i+1)+self.halftab+str(f.split('\\')[-
2])+'\\'+str(f.split('\\')[-1]))
            except:
                self.ui.flufileLW.addItem('#'+str(i+1)+self.halftab+str(f.split('/')[-
2])+'/'+str(f.split('/')[-1]))

    def updateSelectedFluFile(self): #update the selected flu files in the listwidget
        self.fludata = []
        selectedflufiles = self.ui.flufileLW.selectedItems()
        self.selectedflufiles_rows = [self.ui.flufileLW.row(item) for item in selectedflufiles]
        self.selectedflufiles_rows.sort()
        if len(selectedflufiles) != 0:
            try:
                for i, r in enumerate(self.selectedflufiles_rows):
                    data = np.loadtxt(str(self.flufiles[r]),comments='#')
                    for d in data: # replace zero error bar with 10% error
                        if d[2]==0:
                            print('Error bar replaced with 10% of value for entry {}'.format(d))
                            d[2] = float(d[1]) / 10
                    print('\n')

                    self.fludata.append(data)
            except OSError as e:
                print(e)
        self.updateFluPlot()

    def removeFluFile(self): #remove flu files in the listwidget and deselect all flu files in the
listwidget

        to_del = [self.ui.flufileLW.row(item) for item in self.ui.flufileLW.selectedItems()]
        self.flufiles = [f for i,f in enumerate(self.flufiles) if i not in to_del]

        self.ui.flufileLW.clear()
        self.updateFluFile()

    def addFluFitFile(self): #add flu fit files into the listwidget and deselect flu fit files in
the listwidget
        try:
            f, _ = QFileDialog.getOpenFileNames(
                caption = 'Select Multiple Fluorescence Fit Files to import',
                directory = self.directory,
                filter = 'FIT Files (*.fit*; *_fit.txt)'
            )
            self.flufitfiles = self.flufitfiles + f
            self.directory = str(QFileInfo(self.flufitfiles[0]).absolutePath())
            self.updateFluFitFile()
        except IndexError as IE:
            pass # ignore IndexError
        except:
            print("Something went wrong when reading fit files!")
    def updateFluFitFile(self): #update flu fit files in the listwidget
        self.ui.flufitfileLW.clear()
        for i, f in enumerate(self.flufitfiles):
            try:
                self.ui.flufitfileLW.addItem(
                    '#'  +  str(i  +  1)  +  self.halftab  +  str(f.split('\\')[-2])  +  '\\'  +
str(f.split('\\')[-1]))
            except:
                self.ui.flufitfileLW.addItem(
                    '#'  +  str(i  +  1)  +  self.halftab  +  str(f.split('/')[-2])  +  '/'  +
str(f.split('/')[-1]))

    def updateSelectedFluFitFile(self): #update the selected flu fit files in the listwidget
        self.flufitdata = []
        selectedflufitfiles = self.ui.flufitfileLW.selectedItems()
        self.selectedflufitfiles_rows    =    [self.ui.flufitfileLW.row(item)    for    item    in
selectedflufitfiles]
        self.selectedflufitfiles_rows.sort()
        try:
            if len(selectedflufitfiles) != 0:
                for i, r in enumerate(self.selectedflufitfiles_rows):
                    data = np.loadtxt(str(self.flufitfiles[r]),comments='#')
                    self.flufitdata.append(data)
        except OSError as OE:
            print(OE)
        self.updateFluPlot()

    def removeFluFitFile(self):  #remove flu fit files in the listwidget and deselect all flu fit
files in the listwidget

        to_del = [self.ui.flufitfileLW.row(item) for item in self.ui.flufitfileLW.selectedItems()]
        self.flufitfiles = [f for i, f in enumerate(self.flufitfiles) if i not in to_del]

        self.ui.flufitfileLW.clear()
        self.updateFluFitFile()

    def updateFluPlot(self): #update the plot in the flu plotwidget

        ax1 = self.ui.fluPW.canvas.ax
        ax1.clear()

        if self.ui.flulineCB.isChecked():
            ls = '-'
        else:
            ls = ''

        if len(self.fludata) != 0: #plot flu files
            for i,d in enumerate(self.fludata):
                ax1.errorbar(d[:,0],d[:,1],yerr=d[:,2],
                             marker='o', ls=ls,
                             label='#'+str(i+1))

        if len(self.flufitdata) != 0: #plot flu fit files
            for i, d in enumerate(self.flufitdata):
                ax1.plot(d[:, 0], d[:, 1],marker='', ls='-', label=' fit #'+str(i + 1))


        self.xaxis = self.ui.fluxaxisCB.currentText()
        if self.ui.flushowCB.isChecked():
            if np.all(self.flu==0):
                print('Please print simulate button first!!')
                return
            else:
                if self.xaxis == 'Qz':
                    x = self.qz
                    y = self.flu[0, :, 2:]
                    x_range = [x[0]-0.001, x[-1]+0.001]
                    x_label = r'$Q_z$' + ' ' + r'$[\AA^{-1}]$'
                    if self.ui.fluqcCB.isChecked():
                        ax1.axvline(self.qc_true, color='black',ls='--', alpha=0.5)
                        ax1.axvline(self.qc, color='black', alpha=0.5)
                elif self.xaxis == 'Sh':
                    x = self.sh
                    y = self.flu[:, 0, 2:]
                    x_range = [x[0]-0.001, x[-1]+0.001]
                    x_label = r'$\Delta sh$' + ' ' + r'$[mm]$'
            try:
                ax1.set_xlabel(x_label)
                ax1.set_ylabel(r'$Intensity [a.u.]$')
                ax1.set_xlim(x_range)
                ax1.plot(x, y[:,0], ls='-', label='total', color='r')
                if self.ui.flucompCB.isChecked():
                    ax1.plot(x, y[:,1], ls='-', label='water',color='b', alpha=0.5)
                    ax1.plot(x, y[:,2], ls='-', label='interface',color='purple', alpha=0.5)
                    ax1.plot(x, y[:,3], ls='-', label='oil', color='g', alpha=0.5)

            except ValueError as VE:
                print(VE)
        if self.ui.flulegendCB.isChecked():
            ax1.legend(loc = str(self.ui.flulegendlocCoB.currentText()),
                       frameon=False,
                       scatterpoints=0,
                       numpoints=1)
        if self.ui.flugridCB.isChecked():
            ax1.grid(1)
        if self.ui.flulogyCB.isChecked():
            ax1.set_yscale('log')
        else:
            ax1.set_yscale('linear')

        self.ui.fluPW.canvas.draw()

    def setFluPlotScale(self): #set the scale of each data in the flu plot
        if len(self.selectedflufiles_rows)+len(self.selectedflufitfiles_rows)==0:
            self.messageBox('Warning:: No Fluorescence or Fit files selected!')
        else:
            row_flu=len(self.selectedflufiles_rows)
            row_fit=len(self.selectedflufitfiles_rows)
            row=row_flu+row_fit
            Dialog=QDialog(self)
            self.uiplotscale=uic.loadUi(UI_path + 'plotscale.ui', Dialog)
            self.uiplotscale.scaleTW.setRowCount(row) #set the table size; 4 column is fixed
            self.uiplotscale.show()
            self.uiplotscale.scaleLabel.setText('Fluorescence       Plot       Scale       Setup:
X=X*Factor+Offset')
            self.uiplotscale.scaleTW.setHorizontalHeaderLabels(QStringList()<<"X      Factor"<<"X
Offset"<<"Y Factor"<<"Y Offset") #set the horizontal header
            vlabel=QStringList() #set the vertical header
            for i in range(row_flu):
                vlabel.append("Flu #"+str(self.selectedflufiles_rows[i]+1))
            for i in range(row_fit):
                vlabel.append("Fit #"+str(self.selectedflufitfiles_rows[i]+1))
            self.uiplotscale.scaleTW.setVerticalHeaderLabels(vlabel)
            for i in range(row_flu):  #set the initial values
                for j in range(4):

self.uiplotscale.scaleTW.setItem(i,j,QTableWidgetItem(str(self.fluscale[i][j])))
                    self.uiplotscale.scaleTW.item(i,j).setTextAlignment(Qt.AlignCenter)
            for i in range(row_fit):
                for j in range(4):

self.uiplotscale.scaleTW.setItem(i+row_flu,j,QTableWidgetItem(str(self.flufitscale[i][j])))
                    self.uiplotscale.scaleTW.item(i+row_flu,j).setTextAlignment(Qt.AlignCenter)
            self.connect(self.uiplotscale.scaleTW,              SIGNAL('cellChanged(int,int)'),
self.updateFluPlotScale) #update the flu scale and plot
            self.connect(self.uiplotscale.closePB,SIGNAL('clicked()'),       self.closePlotScale)
#close the scale setup window

    def updateFluCal(self): # calculate the flu  based on current parameters.

        self.updatePar()
        # p is the parameter set taken by the core function, must be initialized before updatding
        p = OrderedDict()
        p = fl.update_flu_parameters(p, self.flu_par, self.sys_par,self.flu_elements)
        if not self.ui.flushowCB.isChecked():
            return # if show is not checked, do nothing.
        if self.xaxis == 'Qz':
            self.qc_true = np.sqrt(2*(p['ibDt']-p['itDt'])) * 2 * p['k0']
            self.qc = self.qc_true - p['qoff']
        self.flu = fl.flu2min(self.flu_par, (self.sh, self.qz), p)
        self.updateFluPlot()

    def fitFlu(self, uncertainty_calculation=False):

        selectedflufiles = self.ui.flufileLW.selectedItems()
        try:
            assert len(selectedflufiles) == 1
        except:
            print("Error: please select one data to fit.")
            return
        data = self.fludata[0]

        self.updatePar()

        self.data_to_fit   =   data[(data[:,0]   >=   self.flu_fit_range[0])   *   (data[:,   0]   <=
self.flu_fit_range[1])]

        if self.xaxis == 'Qz':
            self.qz = self.data_to_fit[:,0]
        elif self.xaxis == 'Sh':
            self.sh = self.data_to_fit[:,0]

        # for uncertainty calculation, this function is only used to get preparaed for multicore
processing
        if uncertainty_calculation == True: return

        self.flu_result = lm.minimize(fl.flu2min, self.flu_par,
                                      args=((self.sh, self.qz), self.flucal_par),
                                      kws={'data':self.data_to_fit[:,1],
'eps':self.data_to_fit[:,2]}
                                      )
        self.flu_par = self.flu_result.params

        # for uncertainty calculation, the following is not needed

        tb = self.ui.fluparaTB
        tb.clear()
        tb.append(lm.fit_report(self.flu_result))

        self.updateUI(fresh=False)  # it has to be before updateFluCal()
        self.updateFluCal() # self.updateUI() has to be excucated before it reads parameters from
GUI

    def saveFlu(self):
        if str(self.ui.flusaveCB.currentText())=='Save Fit':
            self.saveFluFitDig()
        elif str(self.ui.flusaveCB.currentText())=='Save Para':
            self.saveFluPara()

    def saveFluPara(self):

        self.updatePar()

        self.saveFileName   =   QFileDialog.getSaveFileName(caption='Save   Fluorescence   Fitting
Parameters',
                                                    directory=self.directory)
        with open(self.saveFileName[0] + '_par.txt','w') as fid:
            try:
                try:
                    fid.write('Chi_Square\t' + format(self.flu_result.redchi, '.3f') + '\n')   #
chisquare
                except:
                    fid.write('Chi_Square\tNA\n')

                fid.write('Fitting_Parameters\n')
                for p, u in self.ui_params.items():
                    fid.write(p + '\t\t' + format(float(u[0].text()),'.3e') + '\n')

                fid.write('\nSystem_Parameters\n')
                fid.write('Beam_Profile\t\t' + self.beam +'\n')
                for p, u in self.ui_syspar.items():
                    fid.write(p + '\t\t' + format(float(u.text()), '.4f') + '\n')
                print("Parameters saved!")

            except:
                print('Oops! Something went wrong, please check your parameters!')

    def loadFlu(self):
        if str(self.ui.fluloadCB.currentText())=='Load Para':
            self.loadFluPara()

    def loadFluPara(self):

        try:
            filename, _ = QFileDialog.getOpenFileName(caption='Select Parameter File to read',
                                                       directory=self.directory,
                                                       filter='Par Files (*.par*;*_par.txt)')
            self.directory = str(QFileInfo(filename).absolutePath())
            with open(str(filename)) as fid:
                fdata=fid.readlines()
        except IOError: # if the dialog is canceled.
            return
        # set ui values with loaded value
        line_num = 0
        line_type = 0 # 0: not a parameter line; 1: fitting parameter line; 2: system parameter
line
        while True:
            try:
                line = fdata[line_num].split()
            except IndexError: # end of file
                break
            if line == []:
                line_type = 0
            else:
                if line[0] == 'Fitting_Parameters':
                    line_type = 1
                elif line[0] == 'Beam_Profile':
                    self.beam = line[1]
                    line_type = 2
                elif line_type == 1:
                    self.flu_par[line[0]].value = float(line[1])
                elif line_type == 2:
                    self.sys_par[line[0]] = float(line[1])
            line_num += 1

        # update parameter with new ui values
        self.updateUI(fresh=False)
        self.updateFluCal()

    def saveFluFitDig(self):

        Dialog=QDialog(self)
        self.uiflusavefit = uic.loadUi(resource_path(UI_path+'refsave.ui'), Dialog)
        self.uiflusavefit.label.setText('Save Fluorescence Fit/Calcualtion!')
        try:
            self.uiflusavefit.xminLE.setText(str(self.flu_simu_range[0]))
            self.uiflusavefit.xmaxLE.setText(str(self.flu_simu_range[1]))
        except:
            pass
        self.uiflusavefit.numpointLE.setText(str(200))

        self.uiflusavefit.cancelPB.clicked.connect(self.cancelSaveFluFit)
        self.uiflusavefit.okPB.clicked.connect(self.saveFluFit)

        self.uiflusavefit.show()

    def cancelSaveFluFit(self):
        self.uiflusavefit.close()
        self.flusavefitindex=0

    def saveFluFit(self):

        try:
            self.flusavefitindex = 1
            self.flunp = float(self.uiflusavefit.numpointLE.text())
            self.fluxmin = float(self.uiflusavefit.xminLE.text())
            self.fluxmax = float(self.uiflusavefit.xmaxLE.text())
            assert (self.fluxmin < self.fluxmax), "Maximum smaller than Minimum"

            self.saveFileName = QFileDialog.getSaveFileName(caption='Save Fluorescence Fit Data',
                                                            directory=self.directory)
            fname = self.saveFileName[0] + '_fit.txt'
            if self.xaxis == 'Qz':
                fit_to_save = self.flu[0,:,(1,2)].transpose()
            elif self.xaxis == 'Sh':
                fit_to_save = self.flu[:,0,(0,2)]
            np.savetxt(fname, fit_to_save, fmt='%.4e\t%.4e')

            self.flusavefitindex=0
            self.uiflusavefit.close()
        except AssertionError as AE:
            print("Error: {0}".format(AE))
        except IndexError as IE:
            pass # Ignore IndexError
        except:
            print("An error happens while saving fit file!")

    def debugErr(self):
        parfile = os.path.join(dir_path_test,
                               'sh_sample03_318_50mMEu(NO3)3_s1h0.2_qz0.0015_par.txt')
        self.updateFluFile()

        with open(str(parfile)) as fid:
            fdata = fid.readlines()
        # set ui values with loaded value
        line_num = 0
        line_type = 0  # 0: not a parameter line; 1: fitting parameter line; 2: system parameter
line
        while True:
            try:
                line = fdata[line_num].split()
            except IndexError:  # end of file
                break
            if line == []:
                line_type = 0
            else:
                if line[0] == 'Fitting_Parameters':
                    line_type = 1
                elif line[0] == 'Beam_Profile':
                    self.beam = line[1]
                    line_type = 2
                elif line_type == 1:
                    self.flu_par[line[0]].value = float(line[1])
                elif line_type == 2:
                    self.sys_par[line[0]] = float(line[1])
            line_num += 1
        # update parameter with new ui values
        self.updateUI()
        self.updateFluCal()

    def fluErrorInit(self):

        # choose the parameter for which the chisq is calculated
        self.fluerr_pname = [] # initialize # of the chosen parameters
        try:
            self.fluerr_pname = [p for p,u in self.ui_params.items() if u[1].isChecked()]
            if len(self.fluerr_pname) != 1:
                raise ValueError
            print("Calculating Chi-square for:", *self.fluerr_pname)
        except ValueError:
            print(" Did u pick the right number of parameters to fit?\n\n")
            # if multiple para's r checked, uncheck all and raise error
            for name in self.fluerr_pname:
                self.ui_params[name][1].setChecked(False)
            return

        self.uifluerr1=uic.loadUi(UI_path + 'err1.ui',QDialog(self))
        self.uifluerr1.label.setText('Uncertainty     Calculation     for     Parameter:'     +
self.fluerr_pname[0])

        best_value = float(self.ui_params[self.fluerr_pname[0]][0].text())
        half_range_to_fit = abs(best_value*0.1)
        # the length of left and right half of range for the chosen values.
        self.uifluerr1.bestvalLE.setText(format(best_value, '.2e'))
        self.uifluerr1.leftLimitLE.setText(  # set left limit
            format(best_value - half_range_to_fit, '.2e'))
        self.uifluerr1.rightLimitLE.setText( # set right limit
            format(best_value + half_range_to_fit, '.2e'))

        self.uifluerr1.numIntervalLE.setText(format(10  ,'d'))

        # connect the pushbutton to next step
        # self.uifluerr1.cancelPB.clicked.connect(lambda x: self.uifluerr1.close())
        self.uifluerr1.cancelPB.clicked.connect(self.uifluerr1.close)
        self.uifluerr1.nextPB.clicked.connect(self.fluErrorPara)
        self.uifluerr1.show()

    def fluErrorPara(self):

        self.uifluerr1.close()
        # calculate a list of values the parameter should take where the chisq is calculated.
        self.fluerr_best_value = float(self.uifluerr1.bestvalLE.text())
        self.fluerr_left_limit = float(self.uifluerr1.leftLimitLE.text())
        self.fluerr_right_limit = float(self.uifluerr1.rightLimitLE.text())
        self.fluerr_num_points = int(self.uifluerr1.numIntervalLE.text())+1
        # append the fittted value for that parameter for displaying that
        # value in the chisq plot as the red dot.
        self.fluerr_fit_range = np.append(self.fluerr_best_value,
                                          np.linspace(self.fluerr_left_limit,
                                                      self.fluerr_right_limit,
                                                      self.fluerr_num_points))
        self.fluerr_chisq_list = np.zeros(self.fluerr_fit_range.shape)

        # automatically toggle the state of fiting and fixed parameters
        for p, u in self.ui_params.items():  u[1].toggle()

        # close the first dialog and open a new dialog
        self.uifluerr2 = uic.loadUi(UI_path + 'err2.ui', QDialog(self))
        self.uifluerr2.label.setText('Please check parameters to fit')
        self.uifluerr2.fluErrorProgress.setValue(0)
        # self.fluErroFit = myThread()
        # self.fluErrorFit.started.connect()
        # self.fluErrorFit.finished.connect(self.fluErrorResult)
        self.uifluerr2.fluErrorProgress.setMaximum(len(self.fluerr_fit_range))
        self.uifluerr2.cancelPB.clicked.connect(self.uifluerr2.close)
        self.uifluerr2.nextPB.clicked.connect(self.fluErrorFit)

        self.uifluerr2.show()

    def fluErrorFitSingleCore(self, q, i, flu_par,flucal_par,data_to_fit):
        fluerr_result = lm.minimize(fl.flu2min, flu_par,
                                    args=((self.sh, self.qz), flucal_par),
                                    kws={'data': data_to_fit[:, 1], 'eps': data_to_fit[:, 2]})
        q.put([i, fluerr_result.nfree, fluerr_result.redchi])



    def fluErrorFit(self):
        self.uifluerr2.label.setText('Calculating the uncertainty for ' + self.fluerr_pname[0])
        self.uifluerr2.nextPB.setEnabled(False) # unable the next push button
        # create a Parameter() object for fitting
        self.updatePar()  # update parameters to fit with GUI

        # make a copy of parameter just for uncertainty calculation
        fluerr_pname = self.fluerr_pname[0]
        fluerr_par = self.flu_par
        fluerr_cal_par = self.flucal_par
        fluerr_par[fluerr_pname].vary = False # make sure the chosen parameter does not vary

        # time the calculation
        start_time = time.time()
        # self.ErrorFit = myThread(self.fitFlu,errorbar=True,args={})



        # self.fitFlu(uncertainty_calculation=True)
        # fluErrorFitSingleCore_i = partial(fluErrorFitSingleCore2,
        #                                   value_list = self.fluerr_fit_range,
        #                                   sh = self.sh,
        #                                   qz = self.qz,
        #                                   pname = fluerr_pname,
        #                                   flu_par = fluerr_par,
        #                                   flucal_par = fluerr_cal_par,
        #                                   data_to_fit = self.data_to_fit)
        #
        #
        # results = multiCore(fluErrorFitSingleCore_i, range(len(self.fluerr_fit_range)))
        # for pp in results: print(pp)
        # print("Uncertainty calculation takes:", time.time() - start_time, "seconds")
        # for result in results:
        #     self.fluerr_chisq_list[result[0]] = result[3]
        # self.fluerr_nfree = results[-1][2]
        # print(self.fluerr_chisq_list)


        self.fitFlu(uncertainty_calculation=True)
        processes = []  # create a pool for processes
        q = multiprocessing.Queue()
        for i,value in enumerate(self.fluerr_fit_range):
            fluerr_par[fluerr_pname].value = value # change value of the chosen parameter
            p = multiprocessing.Process(target=self.fluErrorFitSingleCore,
                                        args=(q, i, fluerr_par,fluerr_cal_par,self.data_to_fit))
            processes.append(p)
            p.start()
        for process in processes:
            process.join()
        results = [q.get(True) for process in processes]

        for pp in results: print(pp)
        print("Uncertainty calculation takes:", time.time()-start_time, "seconds")
        for result in results:
            self.fluerr_chisq_list[result[0]] = result[2]
        self.fluerr_nfree = results[-1][1]
        print(self.fluerr_chisq_list)

        # # fit data and calculate chisq at each grid point
        # for i,para_value in enumerate(self.fluerr_fit_range):
        #     self.fluerr_parameters[self.fluerr_pname_to_fit].value = para_value
        #     fluresult=lm.minimize(self.flu2min, self.fluerr_parameters, args=(x,y,yerr))
        #     self.fluerr_chisq_list[i] = fluresult.redchi
        #     # update progress
        #
        # self.progressDialog.hide()

        self.uifluerr2.close()
        self.fluErrorResult()

    def fluErrorResult(self):
        # calculate the left/right error for the parameter
        funChisqFactor=interp1d(self.errorlist[:,0],self.errorlist[:,1],kind='cubic')
        chisq_factor = funChisqFactor(self.fluerr_nfree) # chisq_factor corresponding to degree of
freedom
        idx_min_chisq = np.argmin(self.fluerr_chisq_list[1:]) + 1
        min_chisq = np.min(self.fluerr_chisq_list[1:])
        self.target_chisq = min_chisq * chisq_factor
        try: # interpolate function of left values against various chisq's
            funChisqListLeft = interp1d(self.fluerr_chisq_list[1:idx_min_chisq+1],
                                        self.fluerr_fit_range[1:idx_min_chisq+1],
                                        kind='linear')
            left_err = self.fluerr_best_value - funChisqListLeft(self.target_chisq)
            left_err_str = format(float(left_err),'.2e')
        except:
            left_err_str = "not found"
        try: # interpolate function of right values against various chisq's
            funChisqListRight = interp1d(self.fluerr_chisq_list[idx_min_chisq:],
                                         self.fluerr_fit_range[idx_min_chisq:],
                                         kind='linear')
            right_err = funChisqListRight(self.target_chisq) - self.fluerr_best_value
            right_err_str = format(float(right_err),'.2e')
        except:
            right_err_str = "not found"

        self.uifluerr3=uic.loadUi(UI_path + 'err3.ui',QDialog(self))
        self.uifluerr3.label.setText('Plot for Chi-square vs Parameter:'+self.fluerr_pname[0])
        self.uifluerr3.minchiLE.setText(format(min_chisq,'.2f'))
        self.uifluerr3.tarchiLE.setText(format(self.target_chisq,'.2f'))
        self.uifluerr3.lefterrLE.setText(left_err_str)
        self.uifluerr3.righterrLE.setText(right_err_str)
        self.uifluerr3.logyCB.stateChanged.connect(self.fluErrorPlot)
        self.uifluerr3.closePB.clicked.connect(lambda x: self.uifluerr3.close())
        self.uifluerr3.closePB.clicked.connect(self.uifluerr3.close)
        self.uifluerr3.savePB.clicked.connect(self.fluErrorSave)
        self.uifluerr3.show()
        self.fluErrorPlot()

    def fluErrorPlot(self):
        the_ax = self.uifluerr3.plotWidget.canvas.ax
        the_ax.clear()
        the_ax.set_xlabel(self.fluerr_pname[0])
        the_ax.set_ylabel('Chi-square')
        # check if y axis is logscale
        if self.uifluerr3.logyCB.checkState()!=0:
            the_ax.set_yscale('log')
        else:
            the_ax.set_yscale('linear')

        # plot the calculated chisq
        the_ax.plot(self.fluerr_fit_range[1:], self.fluerr_chisq_list[1:],
                    marker='o',ls='-')

        # plot the fitted parameter value and corresponding chisq
        the_ax.plot(self.fluerr_fit_range[0], self.fluerr_chisq_list[0],
                    marker='o',color='red')

        # plot the target chisq
        the_ax.plot(self.fluerr_fit_range[[1,-1]],
                    self.target_chisq * np.array([1,1]),
                    ls='-',color='green')

        self.uifluerr3.plotWidget.canvas.draw()

    def fluErrorSave(self):
        print("Save function to be released...")

    def insFluIon(self):  # add one ion in the subphase
        insrows=self.ui.flusubTW.selectionModel().selectedRows()
        insrows=[self.ui.flusubTW.row(self.ui.flusubTW.itemFromIndex(insrows[i]))    for    i    in
range(len(insrows))]
        if len(insrows)!=1:
            self.messageBox('Warning:: Only one row can be seleted!')
        else:
            self.ui.flusubTW.insertRow(insrows[0])
        for i in range(3):
            self.ui.flusubTW.setItem(insrows[0],i,QTableWidgetItem('Cl/2/1.80'.split('/')[i]))

    def rmFluIon(self): #remove one ion in the subphase
        rmrows=self.ui.flusubTW.selectionModel().selectedRows()
        removerows=[]
        for rmrow in rmrows:
            removerows.append(self.ui.flusubTW.row(self.ui.flusubTW.itemFromIndex(rmrow)))
            removerows.sort(reverse=True)
        if len(removerows)==0:
            self.messageBox('Warning:: No ion is selected!!')
        else:
            for i in range(len(removerows)):
                self.ui.flusubTW.removeRow(removerows[i])
```

A.2.7 flu_geometry_2.py

```
import sys
sys.path.append('/Users/zhuzi/work/data_analysis_20190514/')

import flu_geometry_routines as gm

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('xtick',labelsize=20)
mpl.rc('ytick',labelsize=20)
mpl.rc('axes',labelsize=25)
mpl.rc('axes',titlesize=25)
mpl.rc('font',family='serif')
mpl.rc('mathtext',default='regular',fontset='custom')


if __name__ == '__main__':

    sample_height = 0.02121+4.51e-3  # sample height in mm.
    # beam
    E = 20  # Energy in Ke
    qz = 0.006-3.37e-4 # Qz in A^-1
    D = 0.025477  # the thickness of the beam in mm

    # sample cell
    L = 12.7  # the length of the detector area in mm
    gap = 0.5 # the gap between the collamator and interface in mm
    glass_tray = 75.6
    curvature = 215.4  # curvature radius in m
    R_cur = curvature * 1000

    #########################################################


    pt_density = 500  # the number of points drawn with the length of L.
    a = qz / 1.0134 / E
    print("width: {}".format(glass_tray * a))
    F = D / np.sin(a)  # footprint of the beam if the surface is not curved
    l = np.linspace(-L/2, L/2, pt_density+1)
    #   a series of points on surface for plotting
    x_cell = np.linspace(glass_tray/2, -glass_tray/2, pt_density+1)
    surface, _ = gm.surf_points(x_cell,R_cur)

    #########################################################
    fig = plt.figure(figsize=(12,6),dpi=100)
    ax = fig.add_subplot(111)

    # display y_scale in micron.
    ticks_y = mpl.ticker.FuncFormatter(lambda x, pos: '{0:g}'.format(x*1000))
    ax.yaxis.set_major_formatter(ticks_y)
    ax.set_ylim([-0.04,0.06])
    ax.set_xlim([50,-100])
    ax.set_xlabel(r'$X\/(mm)$')
    ax.set_ylabel(r'$Z\/(\mu m)$')

    # axes and detector level height
    ax.axhline(0,color='b',alpha=0.2)
    ax.axvline(0,color='b',alpha=0.2)
    ax.vlines(x=-L/2, ymin=-gap, ymax=gap,linewidth=1, color='b',alpha=0.2)
    ax.vlines(x=L/2, ymin=-gap, ymax=gap,linewidth=1, color='b',alpha=0.2)


    # curved interface and sample edge
    ax.plot(surface[:,0],surface[:,1],linewidth=1,color='r',alpha=1) # curved interface
    ax.vlines(surface[0,0],ymin=-9.8, ymax=surface[0,1], linewidth=1,color='r',alpha=1)
    ax.vlines(surface[-1,0],ymin=-9.8, ymax=surface[-1,1], linewidth=1,color='r',alpha=1)

    # left and right edge of the beam hitting at the surface
    center = [-sample_height/a,0]
    ray_list = [[center[0] - F/2, center[1]],
            center,
            [center[0] + F/2, center[1]]]

    color_list = ['blue','black','green']
    alpha_list = [1, 0.5, 1]
    # X-rays: incident and reflection
    hit_position = np.zeros(3)

    for i, ray in enumerate(ray_list):

        ax.plot(ray[0],ray[1],marker='.',color='black') # intersection of with horizontal
        print(ray)
        hit = gm.hit_surface(ray, -a, R_cur, glass_tray)
        if not np.isinf(hit[1]): # ray hit the sample.
            _, theta = gm.surf_points([hit[0]],R_cur)
            hit_line = np.array([gm.laser_gun(hit, -a, wall=-80),
                                 hit,
                                 gm.laser_gun(hit, a+2*theta, wall=80)])
            ax.plot(hit_line[:, 0], hit_line[:, 1], linewidth=1,
                    color=color_list[i], alpha=alpha_list[i])
        else:  # ray missed sample
            ray_line = np.array([gm.laser_gun(ray, -a, wall=80),
                                 ray,
                                 gm.laser_gun(ray, -a, wall=-80)])
            if np.isinf(hit[0]):  # ray miss the surface
                theta = a
                hit = [R_cur * np.sin(theta), R_cur * (np.cos(theta) - 1)]
                hit_line = np.array([gm.laser_gun(hit, -theta, wall=80),
                                     hit,
                                     gm.laser_gun(hit, -theta, wall=-80)])
                if hit[0] > x_cell[0]: # tangential line beyond sample
                    hit = surface[0]
                    hit_line = np.zeros(hit_line.shape)
                ax.plot(ray_line[:, 0], ray_line[:, 1], linewidth=1,
                        color=color_list[i], alpha=alpha_list[i])
            elif hit[0] > x_cell[0]: # ray hit surface but too high
                hit = surface[0]
                hit_line = np.array([gm.laser_gun(hit, -a, wall=80),
                                     hit,
                                     gm.laser_gun(hit, -a, wall=-80)])
                ax.plot(ray_line[:, 0], ray_line[:, 1], linewidth=1,
                        color=color_list[i], alpha=alpha_list[i])
            else: # ray hit surface but too low
                hit = surface[-1]
                hit_line = np.array([gm.laser_gun(ray, -a, wall=-80),
                                     ray,
                                     gm.laser_gun(ray, -a, wall=x_cell[-1])])
                ax.plot(hit_line[:, 0], hit_line[:, 1], linewidth=1,
                        color=color_list[i], alpha=alpha_list[i])

        hit_position[i] = hit[0]
        ax.plot(hit[0], hit[1], marker='.', color='red')
        print(hit)


   # print out text
    ax.text(-8,-0.021, "Sample Height: %.6fmm" %(sample_height),
            fontsize=15)
    ax.text(-8,-0.028, "Qz:%.4f, Alpha:%.3fmrad" %(qz,a*1000),
            fontsize=15)
    footprint = hit_position[2]-hit_position[0]
    ax.text(-8,-0.035, "Curv:%dm, Footprint: %.2fmm" % (curvature, footprint),
            fontsize=15)

    ax.grid(1)
      plt.show()
```

A.3 Python functions for other calculations

```
# global constants
```

```
PI = np.pi # PI = 3.1415926...
re = 2.818e-5 # classical electron radius, r_e = 2.818e-15 m
N_A = 6.02e23 # Avogadro's number
```

A.3.1 Critical Angle

```
def criticalAngle(rho1,rho2):
    '''
    simple formula to calculate the critical angle between two liquid pahses
    using Qc=4*sqrt(PI*re(rho_bottom - rho_top))
    "J. Phys. Chem. B 2014, 118, 12486-12500" page5, shortly after equation(2).
    Parameters
    ----------
    rho1: electron density with larger value, in A^-3 (0.3346 for water)
    rho2: electron density with smaller value, in A^-3 (0.2595 for dodecane)

    Notes
    -----
    for 3 HNO3: the electron density is 0.362, critical angle: 0.0122
    print criticalAngle(0.0334,0.2596) -> Qc=0.01026 consistant with Wei's
    print criticalAngle(0.3618,0.2596) -> Qc=0.0120
    print criticalAngle(0.348,0.2672) -> Qc=0.0107
    Returns
    -------
    criticalAngle: critical angle between two phases.


    '''
    Qc = 4*np.sqrt(PI*re*(rho1-rho2))
    return Qc
```

A.3.2 Electron density for a solution with given mass density

```
def eDensity(solvent=(18,10),solute=[],mass_density=(1,0)):
    '''
    Parameters
    ----------
    solute: list of solvents with each component in the form of tuple containing
             molecular weight, electrons per molecule and concentration of that component
             e.g. 10e-4 DHDP (546.85,306,1e-4), 0.5M HEH[EHP] (306.4,170,0.5)...
             the mixture of DHDP and HEH[EHP]: [(546.85,306,1e-4),(306.4,170,0.5)]
             Note: concentration in unit of mole/L
    solvent: tuple of the molecular weight and electrons per molecule for the solvent.
            e.g. water (18,10), dodecane(170.34,98)
    mass_density: mass density of the solution and the uncertainty of the measurement , in g/ml

    Notes
    -----
    Calculation, multi-component solutions likewise
    rho_e = (Ne_solute + Ne_solvent)/V # Ne is the number of electrons in the solution
    Ne_solute = N_A * Cons * V * ne_solute # ne is the electrons per mlecule
    Ne_solvent = N_A * (mdens*V-cons*V*mwght1)/mwght2 * ne_solvent
    ==> rho_e = N_A*(cons*ne_solute+(mdens-cons*mwght1)/mwght2*ne_solvent)

    10e-4 DHDP in dodecane: -> 0.2596 A^-3
    eDensity(solvent=(170.34,98),solute=[(546.85,306,1e-4)],mass_density=(0.7495,0))
    3M HNO3 in water:  -> 0.3618 A^-3
    eDensity(solvent=(18,10),solute=[(63,32,3)],mass_density=(1.098,0))
    0.5M HEH[EHP] and 10mM Eu in dodecane:  -> 0.2672

eDensity(solvent=(170.34,98),solute=[(306.4,170,0.5),(151.96,63,0.01)],mass_density=(0.7774,0))
    0.5M citrate in water:  -> 0.348
    eDensity(solvent=(18,10),solute=[(192.12,100,0.5)],mass_density=(1.047,0))
    Return
    ------
    eDensity: electron density of that solution, in A^-3
    '''
    mdens = uf(mass_density[0]*1e-24,mass_density[1]*1e-24)# convert to g/A^3
    # mdens = mass_density * 1e-24  # convert to g/A^3
    c_n = sum([k[1]*k[2]*1e-27 for k in solute])
    c_m = sum([k[0]*k[2]*1e-27 for k in solute])
    rho_e = N_A*(c_n + (mdens-c_m)/solvent[0]*solvent[1])
    return rho_e
```

A.3.3 Electron density for a solution with mass density not given

```
def eDensitySolution(ne=10,r=0.097,C=1,V=None,flag=1):

     '''
     returns the electron density for a given aqueous solution.

     Parameters
     ----------
     ne: number of electrons for each salt "molecule"
     r: ionic radius in solution, see:
        "Marcus, Y. Ionic Radii in Aqueous Solutions. Chem. Rev. 1988, 88, 1475–1498"
     C: concentration of the solution(mol/L)
     rho_0: electron density of pure water.rho_0 = 0.33357A^-3
     rho_w: mass density of pure water. rho_w = 0.997g/ml
     rho_sol: electron density of the solution(A^-3)
     V: partial molor volume of the salt(ml)
     N_A: Avogadro constant. 6.02E23
     1 ml = 1E24 A^3, 1L = 1E27 A^3

     Note:
     -----
     Unit rule:
         volume: ml, electron density: A^-3
     Calculation are based on 1L of water solution. V is the partial volume for 1mol salt,
     so the real partial volume is CV.

     Calculation of partial molar volume (in cm^-3):
         (for 1mole/L solution, sphere packed up)
         v = 2522.5*r^3 (J. Phys. Chem. B 2009, 113, 10285–10291)
         for ErBr3: v = 2522.5*(0.097^3+3*0.198^3) = 61.0441,  consistant with Wei's result.
         for YCl3:  v = 2522.5*(0.097^3+3*0.180^3) = 46.4359
         for SrCl2  v = 2522.5*(0.125^3+2*0.180^3) = 34.3492, consistant with Wei's result.

     electron density of pure water for arbiturary volume V_0
         rho_0 = (rho_w*V_0/18) * N_A * 10  / V_0
               = rho_w*10/18 * N_A
     where 10 is the electron number for water molecule and 18 is the molar mass for water.

     the amount of electrons from water molecules are:
         N_water = (1000-V)*rho_w*10/18 * N_A
     where 10 is the electron number for water molecule and 18 is the molar mass for water.

     the amount of electrons from salt are:
         N_salt = C*ne*N_A

     for C mol/L solution:
         rho_sol = (N_water + N_salt) / 1E27
     plug the equation above you will get:
         rho_sol = rho_0 + (ne*N_A/1E27-V*rho_0/1000)*C

    Return
    ------
    eDensitySolution for flag:
    1: coeffecient for : rho_sol = rho_0 + coeffecient * C
    2: rho for solution: rho_sol
    3: mass density of water part
    4: partial molar volume for the specific ion (1M)

    For testing
    -----------
    V_Cl = eDensitySolution(r=0.180, flag=4)
    V_Y = eDensitySolution(r=0.097, flag=4)
    V_Sr = eDensitySolution(r=0.125,flag=4)
    V_NO3 = eDensitySolution(r=0.177, flag=4)
    V_Eu = eDensitySolution(r=0.106, flga=4)
    YCl3 = V_Y + 3*V_Cl  #--> 46.4359
    SrCl2 = V_Sr + 2*V_Cl #--> 34.3492
    Eu(NO3)3 = V_Eu + 3 * V_NO3 # --> 44.9679

    print eDensitySolution(ne=173,V=61.044, flag=1) #--> 0.08378 for ErBr3
    print eDensitySolution(ne=90,V=46.4359, flag=1) #--> 0.03869 for YCl3
    print eDensitySolution(ne=72,V=34.3492, flag=1) #--> 0.03189 for SrCl2
    print eDensitySolution(ne=156,V=44.9679, flag=1) #--> 0.0789 for Eu(NO3)3
    # edensity for 42mM Eu(NO3)3 is 0.33357 + 0.0789*0.042 = 0.33688
     '''
     rho_0 = 0.33357

     N_A = 6.02E23
     if V==None:
         V = 2522.5 * r**3
     mdensity = (1000-V) * 0.997 / 1000
     coeff = ne*N_A/1E27 - V*rho_0/1000
     rho_sol = rho_0 + coeff * C
     if flag == 1:
         return coeff
     if flag == 2:
         return rho_sol
     if flag == 3:
         return mdensity
     if flag == 4:
         return V
```

A.3.4 Electron density profile calculation

```
def densityProfile(rhos, ds, roughness, rho_0=0.333003, rho_N=0.2574):
    '''
        Takes in the electron density for each layer and calculate the electron
    density profile. The length of the input arrays is the count of sublayers.
    We use same roughness for all interfaces.
        The arguement "rhos" and "ds" themselves only imply layers between two
    phases, i.e., [rho1,rho2,...,rhoN-1],they are then modified to include the
    two bulk phase, resulting "rhos" to be: [rho0,rho1,rho2,...,rhoN], and
    "ds" to be likewise.

    Parameters
    ----------
    rhos: array_like
        An array of electron density of the layers, from water side to oil side.
    ds: array_like
         An array of thickness of the layers, from water side to oil side.
    roughsness: floating number
        roughness which is same for all the interfaces.
    rho_0: floating number
        The electron density of bottom phase, 0.333003 for water.
    rho_N: floating number
        The electron density of upper phase, 0.2574 for dodecane.
    See also
    --------
    Wei, Journal of pyical chamistry B, 2014 Equation(1)

    Returns
    -------
    densityProfile: electron density along z direction.
    '''
    # N is the number of interfaces
    layers = len(rhos)
    N = layers + 1
    # includes upper and bottom phase as first and last layer.
    rhos = np.hstack(([rho_0],rhos,[rho_N]))
    ds = np.hstack(([10000],ds,[10000]))
    # z0 is the position of each interface along z directoin.
    z0 = np.zeros(N)
    for i in range(layers):
        # does not include bulk (ds[0],ds[-1])
        z0[i+1] = z0[i] + ds[i+1]
    # z0[-1] is the sum of the thickness of all the layers.
    d_total = z0[-1]
    # the range of z is 4 times the thickness of the whole interface.
    z = np.arange(-d_total*2,4*d_total,6*d_total/1500) #length=1500 points.
    # calculate rho(z) (Wei, Journal of pyical chamistry B, 2014 Equation(1))
    rho = np.zeros(len(z))
    sqrt_2 = np.sqrt(2)
    for i in range(N):
        x = (z - z0[i])/(sqrt_2*roughness)
        rho = rho - (rhos[i]-rhos[i+1])*errFunction(x)
    rho = 0.5 * (rho + (rho_0+rho_N))

    out = np.vstack((z,rho))
    return out


def errFunction(x):
    '''
        Takes a one dimensional array, gives the error functoin of that array.
    numeric approximation comes from wiki, Abramowitz and Stegun.
    (maximum erroe: 5e-4) (Tested)

    Parameters
    ----------
    x : array_like
        The source array

    See also
    --------
    http://en.wikipedia.org/wiki/Errorfunction#Approximation_with_elementary_functions

    Returns
    -------
    errFunction: ndarray
        The returned array has the same type as "a".

    Examples
    --------

    '''

    a1, a2, a3, a4 = 0.278393, 0.230389, 0.000972, 0.078108
    z = np.zeros(len(x))
    for i,t in enumerate(x):
        if t>0:
            y = 1 + a1*t + a2*t**2 + a3*t**3 + a4*t**4
            z[i] = 1 - 1/y**4
        elif t<0:
            y = 1 - a1*t + a2*t**2 - a3*t**3 + a4*t**4
            z[i] = 1/y**4 - 1
        else: z[i] = 0
    return z
```

A.3.5 Calculate Fresnel Reflectivity

```
def fresnel(q,qc):
    '''
        calculate fresnel reflectivity. (Tested)

    Parameters
    ----------
    q : array_like
        An array of Q's, usually [0,0.02,...,0.5]
    qc : floating number
        Critical angle

    Returns
    -------
    fresnel: an array containing the resnell reflectivity for each q value.
    '''
    q = q * (q>qc) + qc * (q<qc)
    fre = ((q-np.sqrt(q**2-qc**2))/(q+np.sqrt(q**2-qc**2)))**2
    return fre
```

A.3.6 Contribution of given element to absorption coefficient for water solutioin

```
class muElement():
    '''
    calculate the contribution of given element to mu(inverse of attenuation ) in water solution.

    Pamameters
    ----------
    rho_0: the mass density of simple substance of the element, can be found in the database
            in unit (g/cm^3)
    attenul: attenuation length of that element for rho_0 (and a given energy!)
            in unit micron(1 micron=1E-4cm)
    amass: atomic mass for that element
    concentr: concentration in water solution, it is 1M by default.

    Returns
    -------
    muElement: the contribution of that element

    Note
    ----
    The mu, inverse attenuation length, is proportional to massdensity
    mu_0/rho_0 = mu_1/rho_1, thus mu_1 can be calculated

    For testing
    -----------
    Er = muElement(9.05, 21.71, 167.26)
    print "Er:\n", Er
    Y = muElement(4.46, 32.50, 88.906)
    print "Y:\n", Y
    Br = muElement(3.12, 59.76, 79.9, concentr=3)
    print "Br:\n", Br
    Cl = muElement(0.321E-2, 4.2142E5,35.453,concentr=3)
    print "Cl:\n", Cl
    print "mu for ErBr3: %6.2f" %(Er.mu+Br.mu) #--> mu for ErBr3:  21.37
    print "mu for ErCl3: %6.2f" %(Er.mu+Cl.mu) #--> mu for ErCl3:   9.30
    print "mu for YCl3: %6.2f:" %(Y.mu+Cl.mu)  #--> mu for YCl3:   6.921
    '''
    def __init__(self,rho_0,attenul,amass,concentr=1.):
        self.rho_0 = rho_0
        self.rho_1 = 0 # mass density in solution
        self.amass = amass
        self.mu_0 = 10000/attenul # conversion from micron^-1 to cm^-1
        self.mu = 0
        self.concentration = concentr
        self.rho_w = 0
        self.calculate()
    def __str__(self):
        print_str = \
        "atomic mass %6.2f,\
         \nRaw material: mass density %6.3fg/cm^3, mu %6.2fcm^-1, \
         \nCconcentration %4.2fM in water: mass density %6.3fg/cm^3, mu %6.2fcm^-1\n"\
          %(self.amass,self.rho_0,self.mu_0,self.concentration,self.rho_1,self.mu)
        return print_str
    def calculate(self):
        self.rho_1 = float(self.concentration)/1000 * self.amass
        self.mu = self.mu_0 * (self.rho_1/self.rho_0)
```

A.3.7 Interfacial roughness using Capillary Theory

```
def roughness(t,gamma,kappa=1,Qz=0.5,db=0.67,r=4.8,qmax=1.2566,rho=1,flag=1):
    '''
        calculate the interfacial roughness using capillary wave theory.
    Paramerers
    ----------
    t: temperature, in C
    gamma: interfacial tension, in mN/m
    kappa: bending rigidity
    Qz: maximam Qz in reflectivity data, usually ~0.45 in inverse A.
    db: detector acceptence, in Rad. db=v3/L3, v3 is the vertical width of electronic
        slit, while L3 is the distance between sample and the detector. If flag=1,db
        is in unit "mrad", for YAP detector, L3~670. For Pilatus1M detector, each
        pixel is 172um*172um, usually v3=0.172*11=1.892mm; for CCD detector, each
        picxel is 60um*60um, usually v3=0.06*31=1.86mm.
        L3 can be found in "SD Dist" under "Apex" tab in MW_XR software.
    r: average distance between molecules, r=4.8 for DHDP.
    qmax: defaul 2pi/5=1.2566 in Glenn's code, see more in eq1.26 in Mark's book
    rho: mass density difference between lower phase and upper fhase, in g/cm^3
         for system like 1E-7YCl3(water)/1E-4DHDP(dodecane), rho=1-0.78
    flag: choose which equation to use.
    Returns
    -------
    roughness:the interfacial roughness in A.
        flag = 1: reference equation(2) in J.Phys. Chem B 2014, 118, 10662-10674.
        flag = 2: reference eq 3.135 in Mark's book,and in Glenn's mathematica code.
        flag = 3: include kappa(bending regidity in the calculation)
        flag = 4: same as flag=3,except that quadratic term in the integrant is omitted...
    For testing
    -------
    rf = roughness(301-273.5,38,Qz=0.45,db=0.58e-3,r=4.8,flag=1) #->3.72, data in the reference,
tested.
    r = roughness(25,51.5,Qz=0.25, db=0.6/676, rho=1-0.78,flag=2) #->3.4452, data in Glenn's code,
tested.
    r     =     roughness(25,25.2,Qz=0.2,     db=0.3/638,     rho=1.203-1,flag=2)     #->5.1466,
nitrobenzene/water,tested.
    r = roughness(23.3,47.52,kappa=1,qmax=2*np.pi/8.9,Qz=0.5,db=0.8/672,rho=1.01,flag=3)
        #->3.0987, (3.0833 in Glenn's code:roughness_bending_rigidity.nb),tested.
    r = roughness(23.3,47.52,kappa=1,qmax=2*np.pi/8.9,Qz=0.5,db=0.8/672,rho=1.01,flag=4)
        #->3.3820, (3.2632 in Glenn's code:roughness_bending_rigidity.nb),tested.
    r = roughness(22.5,17.86,kappa=1,Qz=0.3,db=1.0/1695,r=4.8,rho=1.047-0.7774,flag=2)
    r = roughness(22,45.84,Qz=0.38,db=1.892/2745,flag=1) #->3.38, (0.5Mcitrate/dodecane)
    '''
    kBT = 1.38e-23 * (273.15+t) * 1e7 #1e7: convert from N*m into dyne*cm
    PI = np.pi # pi=3.1415926...
    qmax = 2*PI/5 # qmax = 2pi/5=1.2566... see more in eq 1.26 in Mark's book
    g = 981.0 # gravitational acceleration in cm/s^2
    qg = np.sqrt(g*rho/gamma)*1e-8 # inverse capillary length in A^-1
    C = kBT/(2*PI*gamma) * 1e16 # some coefficient; 1e16:convert cm^2 into A^2
    if flag==1:
        sigma_square = C * np.log(4/(Qz*db*r))
    elif flag==2:
        sigma_square = C * np.log(qmax/(Qz*db/4+np.sqrt(qg**2+(Qz*db/4)**2)))
    elif flag==3:
        q2 = gamma/kappa/kBT*1e-16 # in A^-2
        integrant = lambda q: q/(q**2+qg**2+q**4/q2)
        sigma_square  =  C  *  integrate.quad(integrant,Qz*db/2,qmax)[0]  #  quad  returns  a  tuple:
(result,error)
    elif flag==4:
        q2 = gamma/kappa/kBT*1e-16
        integrant = lambda q: q/(q**2+qg**2)
        sigma_square  =  C  *  integrate.quad(integrant,Qz*db/2,qmax)[0]  #  quad  returns  a  tuple:
(result,error)
    return np.sqrt(sigma_square)
```

A.3.8 <u>Ratio of reflectivity to Fresnel reflectivity R/R_F</u>

```
def RRF(q, ref, err, q_off=0, qc=0.0103):
    '''
        Cauculate the ratio R/R_f for a given set of data. (Tested)

    Parameters
    ----------
    q : array_like
       An array of Q's, usually [0,0.02,...,0.5]
    q_off: floating number
        q offset.
    qc : floating number
        Critical angle, is 0.0103 for water and dodecane interface.
    ref: array_like
        an array of raw reflectivity data for each q value.
    err: array_like
        an array of error for each raw reflectivity data

    Returns
    -------
    RRF: ndarray
        Returns a 2-D array with rows: q+q_off, ref/frsnell, err/fresnell
        '''
    # convert input into nparray, q_off set included ->
    (q,ref,err) = map(np.array, (q+q_off,ref,err))
    #calculate fresnel reflectivity ->
    frs = fresnel(q,qc)
    # calculate the ratio for signal and error ->
    refFresnelRatio = np.divide(ref,frs)
    errFresnelRatio = np.divide(err,frs)
    # pack the data into 3 columns ->
    out = np.vstack((q,refFresnelRatio,errFresnelRatio))
    out = np.transpose(out)
    return out

def RtoRRf(openfile, q_off=0, qc=0.0103, save=None):
    '''
        Read reflectivity raw data from a file, and convert it to R/Rf. The
    return value of this function is optimized for saving the result directly
    into a file (see more in Reurns).

    Parameters
    ----------
    openfile: the file containing the raw data
    q_off: q offset is zero if not specified.
    q_c: critical qz is 0.0103 if not specified.
    save: (if specified) the file to which converted data is saved.

    Returns
    -------
    convert: always return a 2-D array with columns
             qz+q_off, R/Rf, err/Rf
    '''
    #load the raw reflectivity data ->
    ref_data = np.loadtxt(openfile)

    # split the raw data into three numpy arrays ->
    q = ref_data[:,0]
    ref = ref_data[:,1]
    err = ref_data[:,2]
    # calculate R/Rf, and transpose it into columns(q,ref,err) ->
    R_Rf = np.transpose(RRF(q,ref,err,q_off=q_off,qc=qc))
    print("data converted with q_off=%6.4f,qc=%6.4f" %(q_off,qc))
    # save R_Rf to file ->
    if save != None:
        np.savetxt(save, R_Rf,fmt="%.4f\t%.8f\t%.8f")
        print("Saved to file:", save)
    return R_Rf
```

Appendix B

Liberalization of PNAS copyright policy: Noncommercial use freely allowed

(PNAS August 24, 2004 101 (34) 12399)

We have changed our copyright and permissions policies to make it easier for authors and readers to use material published in PNAS for research or teaching. Our guiding principle is that, while PNAS retains copyright, anyone can make noncommercial use of work in PNAS without asking our permission, provided that the original source is cited. For commercial use (e.g., in books for sale or in corporate marketing materials), we approve requests on an individual basis and may ask for compensation. We have revised our copyright assignment form to make the changes clear (www.pnas.org/misc/copyright.pdf) and added to our web site a "frequently asked questions" (FAQ) section on author and reader rights (www.pnas.org/misc/authorfaq.shtml).

As a PNAS author, you automatically have the right to do the following:

1. Post a PDF of your article on your web site.
2. Post a webcast containing material from your article.
3. Make electronic or hard copies of articles for your personal use, including classroom use.
4. Use, after publication, all or part of your article in a printed compilation of your work, such as collected writings or lecture notes.
5. Include your article in your thesis or dissertation.
6. Reuse your original figures or tables in your future works.
7. Post a preprint of your article on a public electronic server, provided that you do not use the files created by PNAS.
8. Present your paper at a meeting or conference, including those that are webcast, and give copies of your paper to meeting attendees before or after publication in PNAS. For

interactions with the media prior to publication, see the PNAS policy on media coverage (www.pnas.org/misc/forms.shtml).

9. Permit others to use your original figures or tables published in PNAS for noncommercial use (e.g., in a review article), provided that the source is cited. Third parties need not request permission to use figures and tables for such use.

Given that authors and readers can automatically use original material in PNAS for research or teaching, why do we request copyright transfer? We do so for three reasons: to allow us to publish, archive, and migrate articles to new media; to remove the administrative burden of rights and permissions management from authors; and to provide protection from copyright abuse.

We do not feel that this or any copyright policy is the only one possible. In fact, our policy has changed through our 90 years of publishing and surely will change again. We have requested that authors transfer copyright only since 1993. From the first issue of PNAS in 1915 through 1992, authors held copyright to their articles. From 1978 to 1992, we registered copyright for each journal issue as a collected work but did not request copyright for individual articles. In 1993, we began requiring that authors transfer copyright "in all forms, languages, and media now or hereafter known," which granted us the rights to publish papers online in 1997 and to then digitize selected back issues and post them online.

We think that our current policy best meets the needs of readers, authors, and the journal, for the following reasons:

1. **To store and migrate archival formats of the journal.** We are committed to facilitating permanent, freely accessible archives of the scientific literature. PNAS is a charter member of PubMed Central, a digital archive of the life sciences journal literature (www.pubmedcentral.nih.gov), and is a participant in the National Library of Medicine's

effort to digitize and post back issues of journals. Not holding copyright to individual articles from 1915 to 1992 delayed our posting of this older material online because we do not have the legal rights to do so. In the end we proceeded without explicit permission from the original authors or their heirs. We accept the risk in doing so because we believe it is clearly in everyone's best interest. If a copyright holder objects, however, we will immediately remove the article from our online collection. Full copyright transfer allows publishers explicit rights to invest in long-term archiving strategies.

2. **To provide an administrative convenience for everyone.** Despite our liberal rights and permissions policies, PNAS still receives more than 50 commercial and noncommercial permission requests per week. We routinely agree to noncommercial use, so such requests waste everyone's time.

Unfortunately, PNAS cannot provide permission for others to use all or part of articles published from 1915 to 1992 because we do not hold copyright. Only the original authors or their designees can grant permission. Researchers are frustrated when they contact us for permission to use seminal works and we are unable to grant their requests.

3. **To provide international protection regarding infringement or plagiarism.** On the rare occasion that material is misused, authors appeal to PNAS to intervene on their behalf to enforce copyright protection. In such cases, a formal query from PNAS or the threat of a copyright infringement lawsuit has prompted expeditious action. In cases of redundant publication we sanction authors for violating journal and copyright policy. Because international standards and copyright law are complex, PNAS leaves interpretation of global copyright standards to our expert legal counsel.

We also support creative efforts such as charting, mining, analyzing, sorting, navigating, and displaying information contained in PNAS. The highly successful Sackler Colloquium “Mapping Knowledge Domains” (www.pnas.org/content/vol101/suppl_1) is a prime example (**1**). We encourage authors to use standard forms of data presentation to facilitate this process.

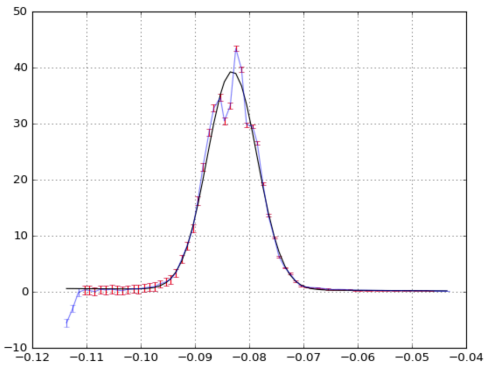
Appendix C

C.1 <u>Beam Profile measurement. Beamtime: 2019 July; beamline: APS15-IDC.</u>

A 4 mm thick and round-edged W blade was mounted on the sample stage — a position where the beam would otherwise hit the sample. The blade was housed in JJ slits slot with its edge facing up. A sample height scan collects the X-ray intensities at 100 different sample height positions from where the blade completely misses the beam to fully block the beam. Deconvoluting the intensities with the blade shape yields the real beam profile.

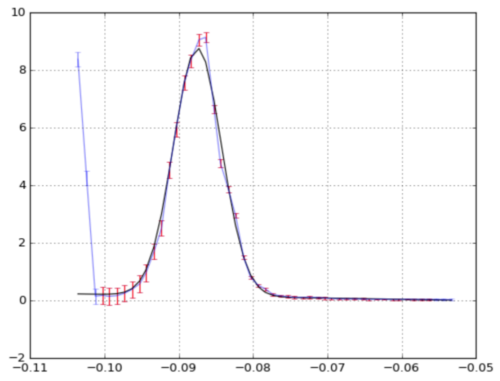
Below is the deconvoluted beam profile with CRL lens. Spec file: pre alignment, not part of 20190718. Scan 124 was done with “s1h 1 1” and scan 125 with “s1h 0.1 0.1”. The resulted beam profile was fitted to a gaussian function. Bigger horizontal opening of slit 1 yields bigger FWHM of the beam size in vertical dimension.

Scan 124 in spec file 20190717. sigma=4.6 µm



```
[[Variables]]
    li_slope:      -5.849 (fixed)
    li_intercept:  -0.159 (fixed)
    g1_sigma:       0.00460932 +/- 6.98e-05 (1.51%) (init= 0.01)
    g1_center:     -0.08307549 +/- 0.000100 (0.12%) (init=-0.08)
    g1_amplitude:   0.44997849 +/- 0.008039 (1.79%) (init= 1)
    g1_fwhm:        0.01085413 +/- 0.000164 (1.51%)  == '2.3548200*g1_
    g1_height:      38.9461476 +/- 0.652280 (1.67%)  == '0.3989423*g1_
[[Correlations]] (unreported correlations are <  0.100)
    C(g1_center, g1_amplitude)   = -0.684
    C(g1_sigma, g1_center)       = -0.664
    C(g1_sigma, g1_amplitude)    =  0.495
```

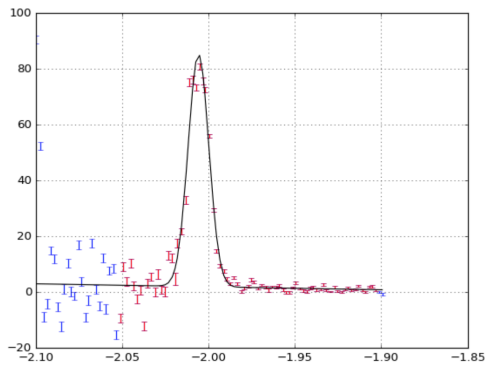
Scan 125 in spec file 20190717. sigma=3.2 µm



```
[[Variables]]
    li_slope:       -4.3246 (fixed)
    li_intercept:   -0.2364 (fixed)
    g1_sigma:        0.00324181 +/- 5.32e-05 (1.64%) (init= 0.01)
    g1_center:      -0.08746799 +/- 7.35e-05 (0.08%) (init=-0.08)
    g1_amplitude:    0.06995733 +/- 0.001309 (1.87%) (init= 1)
    g1_fwhm:         0.00763389 +/- 0.000125 (1.64%)  == '2.3548200*g
    g1_height:       8.60904478 +/- 0.151226 (1.76%)  == '0.3989423*g
[[Correlations]] (unreported correlations are <  0.100)
    C(g1_center, g1_amplitude)   = -0.671
    C(g1_sigma, g1_center)       = -0.631
    C(g1_sigma, g1_amplitude)    =  0.506
```

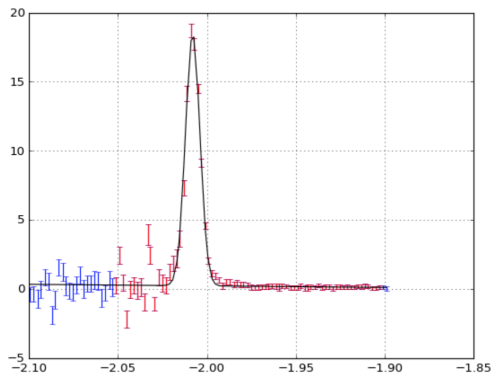
Below is the deconvoluted beam profile with MRO lens. Spec file: 20190718. Scan 287 was done with “s1h 1 1” and scan 288 with “s1h 0.1 0.1”. The resulted beam profile was fitted to a gaussian function. Bigger horizontal opening of slit 1 yields bigger FWHM of the beam size in vertical dimension. Smaller slit size “s1h 0.1 0.1” was used for measurement. The FWHM is 9.6 µm. Before 201907 beamtime, the wider slit, therefore the wider beam with FWHM=15 µm, was used; after 201907 beamtime, FWHM=10 µm.

Scan 287 in spec file 20190718. sigma=5.97µm



```
[[Variables]]
    li_slope:       -10.727 (fixed)
    li_intercept:   -19.724 (fixed)
    g1_sigma:        0.00597738 +/- 0.000164 (2.75%) (init= 0.01)
    g1_center:      -2.00577954 +/- 0.000188 (0.01%) (init=-2.01)
    g1_amplitude:    1.24848658 +/- 0.033248 (2.66%) (init= 2)
    g1_fwhm:         0.01407566 +/- 0.000387 (2.75%)  == '2.3548200
    g1_height:       83.3264312 +/- 2.227261 (2.67%)  == '0.3989423
[[Correlations]] (unreported correlations are <  0.100)
    C(g1_center, g1_amplitude)   = -0.551
    C(g1_sigma, g1_amplitude)    =  0.513
    C(g1_sigma, g1_center)       = -0.477
```

Scan 288 in spec file 20190718. sigma=4.09µm



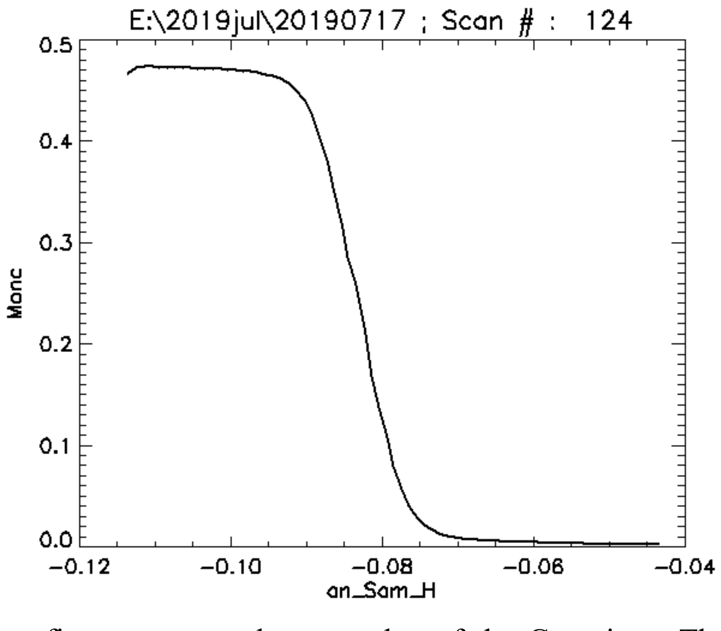
```
[[Variables]]
    li_slope:      -1.141333 (fixed)
    li_intercept:  -2.073334 (fixed)
    g1_sigma:       0.00408824 +/- 8.90e-05 (2.18%) (init= 0.01)
    g1_center:     -2.00798703 +/- 0.000101 (0.01%) (init=-2.01)
    g1_amplitude:   0.18666848 +/- 0.003905 (2.09%) (init= 0.5)
    g1_fwhm:        0.00962708 +/- 0.000209 (2.18%)  == '2.3548200*g1_s
    g1_height:      18.2156205 +/- 0.371600 (2.04%)  == '0.3989423*g1_a
[[Correlations]] (unreported correlations are <  0.100)
    C(g1_sigma, g1_amplitude)    =  0.544
    C(g1_center, g1_amplitude)   = -0.537
    C(g1_sigma, g1_center)       = -0.453
```

C.2 Fits for CRL lens

The data is from SPEC file 20190717.

Data is obtained by scanning a slit edge across the beam and recording transmitted intensity, normalized to incoming beam intensity. Typical data shown in Fig. 1

Assuming the beam profile is Gaussian, the result of the scan is reversed error function, possibly combined with some background. Therefore, fitting to error function (with linear background)

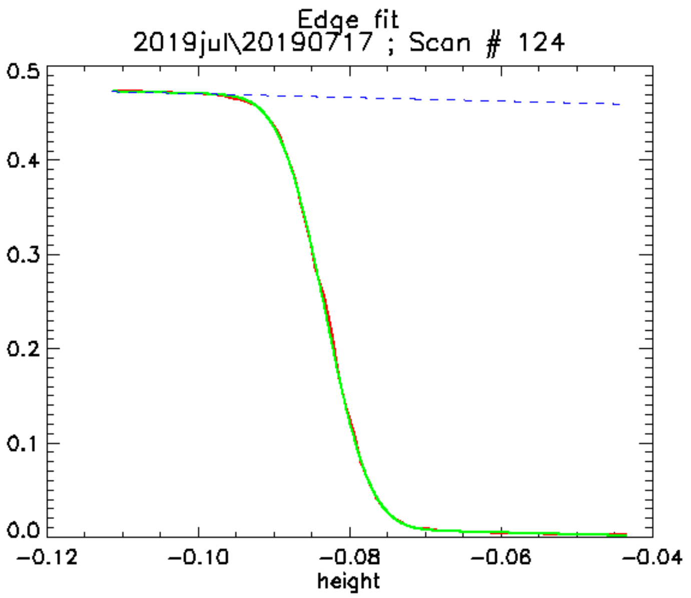


will yield, among other fit parameters, the σ - value of the Gaussian. The FWHM value (if required) is obtained multiplying σ by $\sqrt{\ln(256)} \approx 2.355$.

Alternatively, we can perform numerical differentiation of the data above, which will yield an (inverted) Gaussian and then fit the result to a Gaussian. In such case a correction is needed since numerical differentiation using a finite size step introduces some broadening. The correction is straightforward, though, and following the correction the results of both types of fits are, within margin of error, the same. In general, though, fitting to error function, with no differentiation, is preferable, since numeric differentiation is susceptible to noise.

The fit results are shown on the next page. The red curve represents the data, the green curve is the fit. Usually little of the red curve is visible since the fit covers it. For perfect fit it would cover it perfectly.

Scan #124:

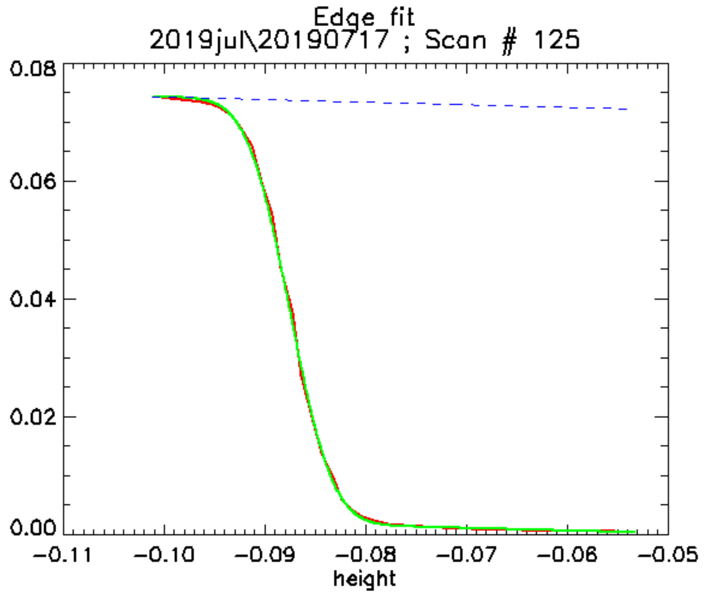


Fit results (error values in parentheses):

$$\sigma = 4.700\mu \quad (0.033\mu)$$

$$\text{FWHM} = 11.068\mu \quad (0.078\mu)$$

Scan #125:



Fit results (error values in parentheses):

$$\sigma = 3.444\mu \quad (0.053\mu)$$
$$\text{FWHM} = 7.875\mu \quad (0.126\mu)$$

Appendix D

Data for HDEHP and [HEH]EHP measurement.

All data are stored in:

Box/research_group_mark_box/Group Members/Zhu/data/

Folders for different beamtime:

2018 November beamtime: 201811Nov/; 2019 April beamtime: 201904Apr/;

2019 July beamtime: 201907Jul/; 2019 December beamtime: 201912Dec/.

Data with each scan are named with the corresponding scan number, the sample it belongs to and the composition of the sample, for the sake of quick reference. All the data associated with the same scan number have the same file name, except for the suffix corresponding to different data type. Data files end with "_flu.txt"; parameter files end with "_par.txt"; fitted curve end with "_fit.txt".

D.1 Other important data for sample height scan on HDEHP sample

D.1.1 2019 December

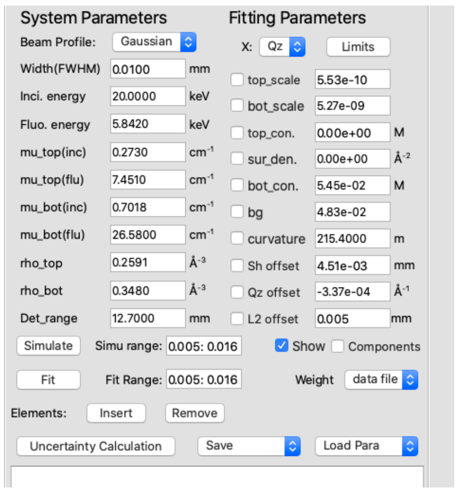
In this beamtime we measured samples with DTPA added in the aqueous phase. All samples have 1mM Eu/10mM HDEHP in dodecane as the organic phase. All aqueous phases contain 0.1mM DTPA, except for pure water, in addition to the complexant. Sample details are listed in the table below:

Sample ID	Oil Phase	Water phase	Notes
3	Pure dodecane	50mM Eu(NO3)3	Water Calibration
4	1mM Eu 10mM HDEHP (organic phase unclean, filtered oil solution twice for sample 6)	HNO3/0.1mM DTPA pH=3	Nitrate concentration: 0.55mM
5		Pure water	Oil Calibration
6		Pure water	Oil Calibration
7	1mM Eu 10mM HDEHP (new oil solution)	Citric/0.1mM DTPA pH=3	Citric concentration: 0.95mM
8		HNO3/0.1mM DTPA pH=3	Nitrate concentration: 0.55mM
9		0.5M Citric/0.1mM DTPA pH=3	Repeat sample 7 (0.5M citrate)

10		0.5M Citric/0.1mM DTPA pH=3	Repeat sample 9 (0.5M citrate)

Sample 3: 50 mM Eu(NO$_3$)$_3$ in water / pure dodecane

This sample is calibration sample to obtain the calibration factor for XFNTR measurement.



System Parameters

Beam Profile: Gaussian

Width(FWHM) 0.0100 mm

Inci. energy 20.0000 keV

Fluo. energy 5.8420 keV

mu_top(inc) 0.2730 cm^{-1}

mu_top(flu) 7.4510 cm^{-1}

mu_bot(inc) 0.7018 cm^{-1}

mu_bot(flu) 26.5800 cm^{-1}

rho_top 0.2591 Å^{-3}

rho_bot 0.3480 Å^{-3}

Det_range 12.7000 mm

Fitting Parameters

X: Qz

top_scale 5.53e-10

bot_scale 5.27e-09

top_con. 0.00e+00 M

sur_den. 0.00e+00 Å^{-2}

bot_con. 5.45e-02 M

bg 4.83e-02

curvature 215.4000 m

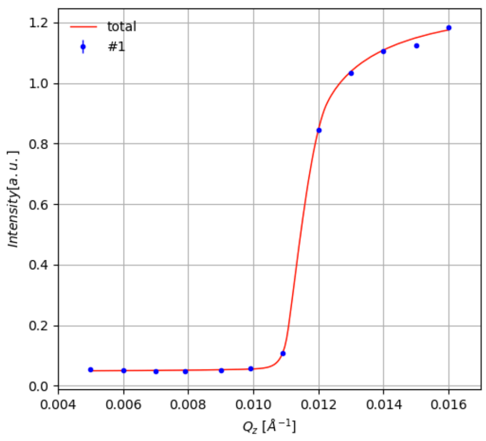
Sh offset 4.51e-03 mm

Qz offset -3.37e-04 Å^{-1}

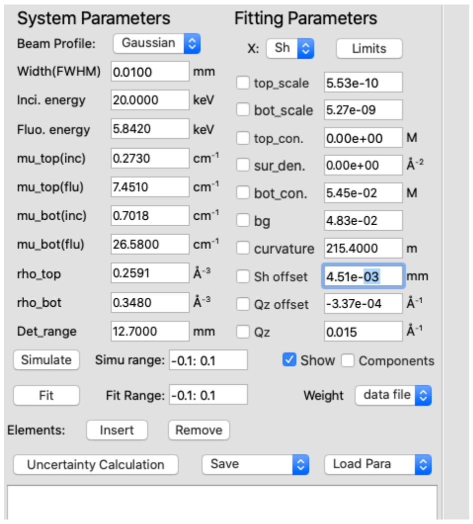
L2 offset 0.005 mm

Simu range: 0.005: 0.016 ✓ Show Components

Fit Range: 0.005: 0.016 Weight data file



We also performed a sample height scan at Q$_z$ = 0.015 Å^{-1} (blue dots, scan number 315) Q$_z$ = 0.006 Å^{-1} (green dots, scan number 309)



System Parameters

Beam Profile: Gaussian

Width(FWHM) 0.0100 mm

Inci. energy 20.0000 keV

Fluo. energy 5.8420 keV

mu_top(inc) 0.2730 cm^{-1}

mu_top(flu) 7.4510 cm^{-1}

mu_bot(inc) 0.7018 cm^{-1}

mu_bot(flu) 26.5800 cm^{-1}

rho_top 0.2591 Å^{-3}

rho_bot 0.3480 Å^{-3}

Det_range 12.7000 mm

Fitting Parameters

X: Sh

top_scale 5.53e-10

bot_scale 5.27e-09

top_con. 0.00e+00 M

sur_den. 0.00e+00 Å^{-2}

bot_con. 5.45e-02 M

bg 4.83e-02

curvature 215.4000 m

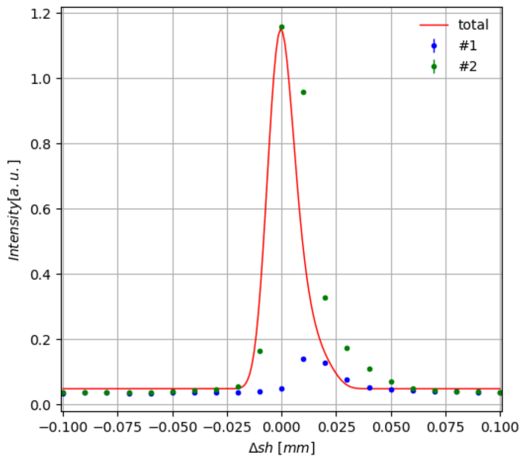
Sh offset 4.51e-03 mm

Qz offset -3.37e-04 Å^{-1}

Qz 0.015 Å^{-1}

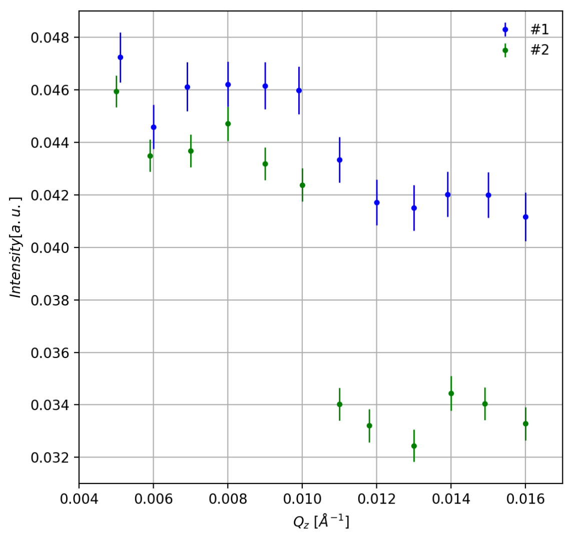
Simu range: -0.1: 0.1 ✓ Show Components

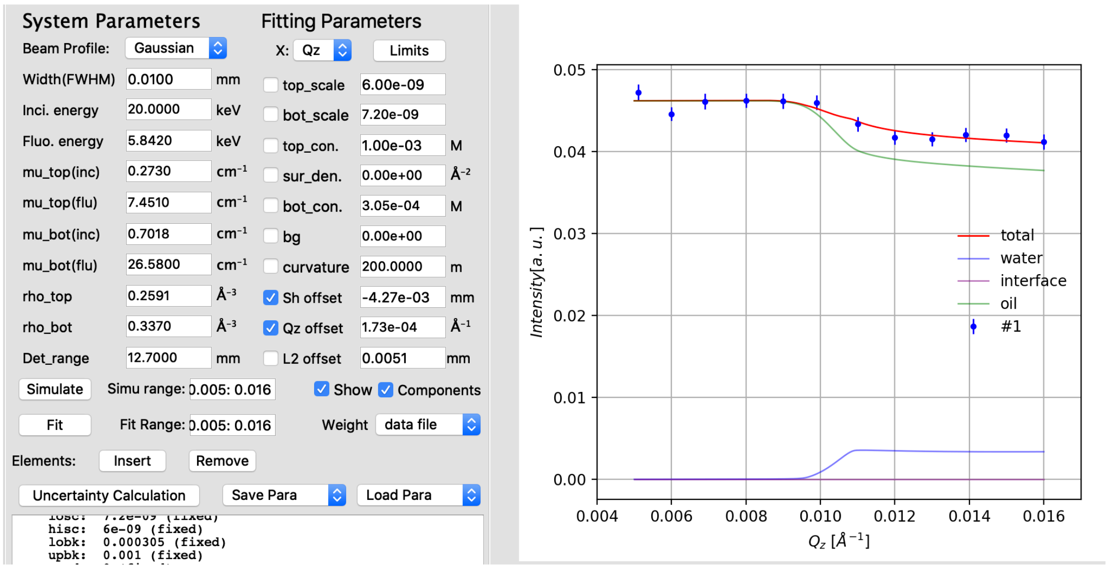
Fit Range: -0.1: 0.1 Weight data file



Sample 4: 1 mM Eu, 10 mM HDEHP in dodecane / 0.55mM HNO$_3$, 0.1 mM DTPA pH=3 in water

585 was at nominal sample height position, #595 repeated #585 by shifting the sample height position up by 6.8 µm. Clearly, the offset of sample height has an big impact on data above critical angle due to the change of the fluorescence volume in aqueous phase.





System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0100	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	$Å^{-3}$
rho_bot	0.3370	$Å^{-3}$
Det_range	12.7000	mm

Fitting Parameters

X: Qz

top_scale	6.00e-09	
bot_scale	7.20e-09	
top_con.	1.00e-03	M
sur_den.	0.00e+00	$Å^{-2}$
bot_con.	3.05e-04	M
bg	0.00e+00	
curvature	200.0000	m
✓ Sh offset	-4.27e-03	mm
✓ Qz offset	1.73e-04	$Å^{-1}$
L2 offset	0.0051	mm

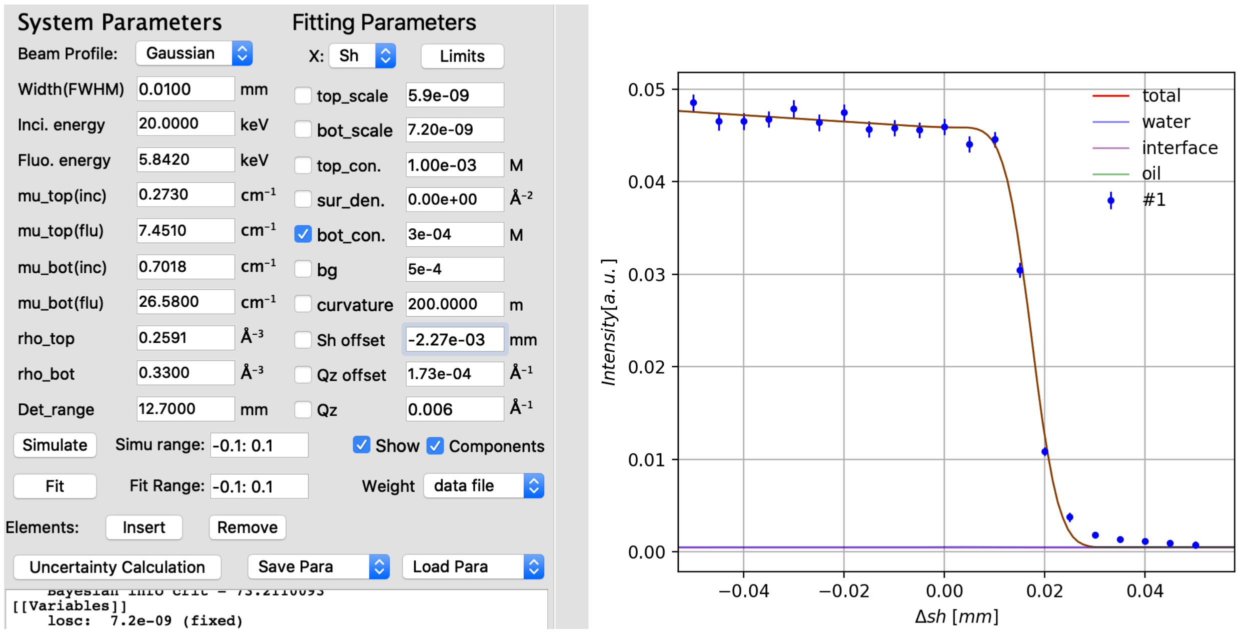
Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight data file

Elements:

```
    losc:   7.2e-09 (fixed)
    hisc:   6e-09 (fixed)
    lobk:   0.000305 (fixed)
    upbk:   0.001 (fixed)
```

Sample height shscan was done at Q_z = 0.006 $Å^{-1}$ (top: #587) and Q_z = 0.015 $Å^{-1}$ (Bottom: #586 & #592). Sh-scan above critical angle confirm the amount of Eu ions in aqueous phase.


System Parameters

Beam Profile: Gaussian

Width(FWHM) 0.0100 mm

Inci. energy 20.0000 keV

Fluo. energy 5.8420 keV

mu_top(inc) 0.2730 cm^{-1}

mu_top(flu) 7.4510 cm^{-1}

mu_bot(inc) 0.7018 cm^{-1}

mu_bot(flu) 26.5800 cm^{-1}

rho_top 0.2591 Å^{-3}

rho_bot 0.3300 Å^{-3}

Det_range 12.7000 mm

Simu range: -0.1: 0.1

Fit Range: -0.1: 0.1

Fitting Parameters

X: Sh

☐ top_scale 5.9e-09

☐ bot_scale 7.20e-09

☐ top_con. 1.00e-03 M

☐ sur_den. 0.00e+00 Å^{-2}

☑ bot_con. 3e-04 M

☐ bg 5e-4

☐ curvature 200.0000 m

☐ Sh offset -2.27e-03 mm

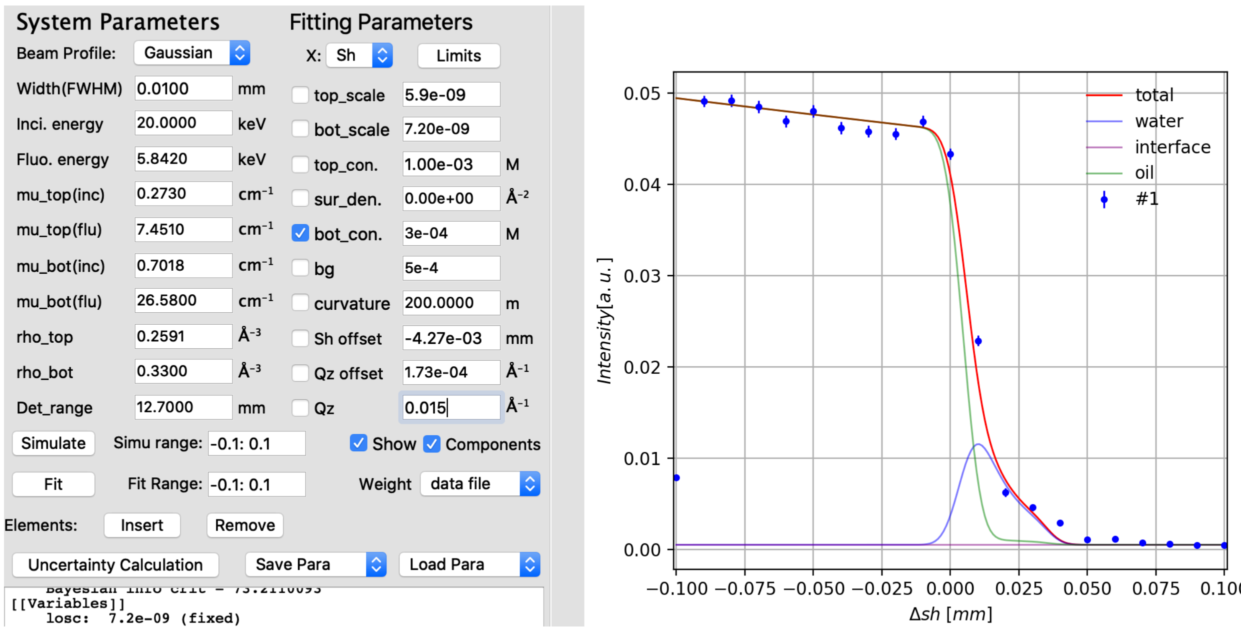
☐ Qz offset 1.73e-04 Å^{-1}

☐ Qz 0.006 Å^{-1}

☑ Show ☑ Components

Weight data file

```
    Bayesian info crit = 73.2110093
[[Variables]]
    losc:   7.2e-09 (fixed)
```


System Parameters

Beam Profile: Gaussian

Width(FWHM) 0.0100 mm

Inci. energy 20.0000 keV

Fluo. energy 5.8420 keV

mu_top(inc) 0.2730 cm^{-1}

mu_top(flu) 7.4510 cm^{-1}

mu_bot(inc) 0.7018 cm^{-1}

mu_bot(flu) 26.5800 cm^{-1}

rho_top 0.2591 Å^{-3}

rho_bot 0.3300 Å^{-3}

Det_range 12.7000 mm

Simu range: -0.1: 0.1

Fit Range: -0.1: 0.1

Fitting Parameters

X: Sh

☐ top_scale 5.9e-09

☐ bot_scale 7.20e-09

☐ top_con. 1.00e-03 M

☐ sur_den. 0.00e+00 Å^{-2}

☑ bot_con. 3e-04 M

☐ bg 5e-4

☐ curvature 200.0000 m

☐ Sh offset -4.27e-03 mm

☐ Qz offset 1.73e-04 Å^{-1}

☐ Qz 0.015 Å^{-1}

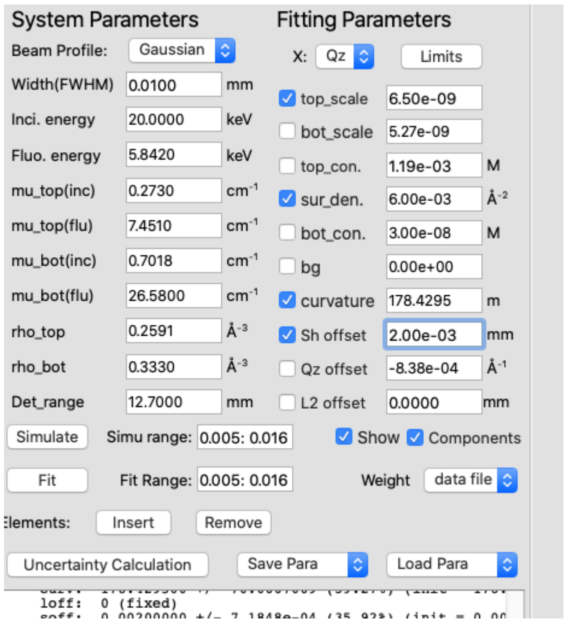
☑ Show ☑ Components

Weight data file

```
    Bayesian info crit = 73.2110093
[[Variables]]
    losc:   7.2e-09 (fixed)
```

Sample 6. 1 mM Eu, 10 mM HDEHP / pure water

This Qz-scan was also used to obtain the oil calibration factor. Scan number: 648. The non-zero interfacial ion distribution is later confirmed by sh-scan.



System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0100	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	$Å^{-3}$
rho_bot	0.3330	$Å^{-3}$
Det_range	12.7000	mm

Fitting Parameters

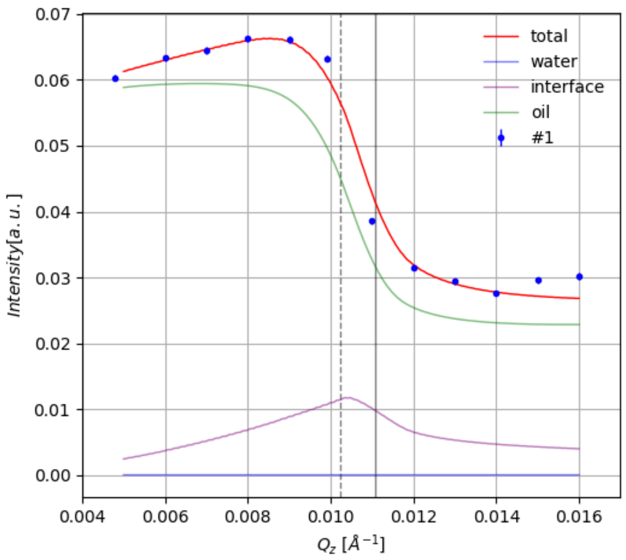
X: Qz

✓	top_scale	6.50e-09	
	bot_scale	5.27e-09	
	top_con.	1.19e-03	M
✓	sur_den.	6.00e-03	$Å^{-2}$
	bot_con.	3.00e-08	M
	bg	0.00e+00	
✓	curvature	178.4295	m
✓	Sh offset	2.00e-03	mm
	Qz offset	-8.38e-04	$Å^{-1}$
	L2 offset	0.0000	mm

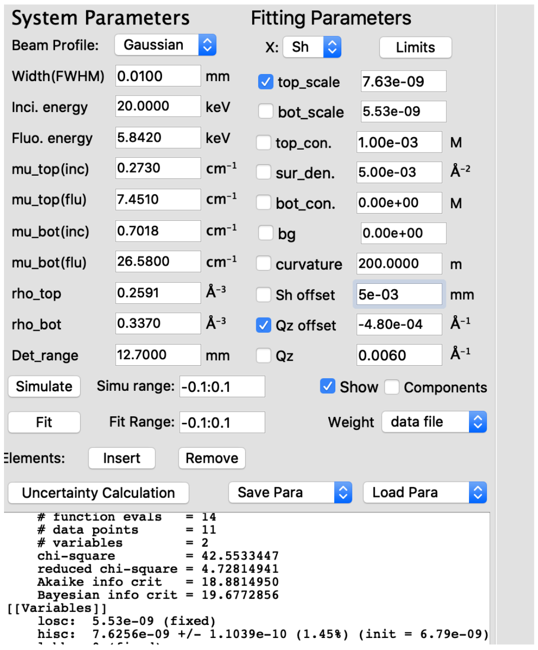
Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight: data file

```
    loff:  0 (fixed)
```



Below are a set of sample height scans at $Q_z = 0.006$ $Å^{-1}$ with best fitting on top. Scan numbers: #640, 641, 649, 651. The small bump at the edge of the fall indicates non zero distribution of ions in the evanescent wave region below the interface.



System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0100	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	$Å^{-3}$
rho_bot	0.3370	$Å^{-3}$
Det_range	12.7000	mm

Fitting Parameters

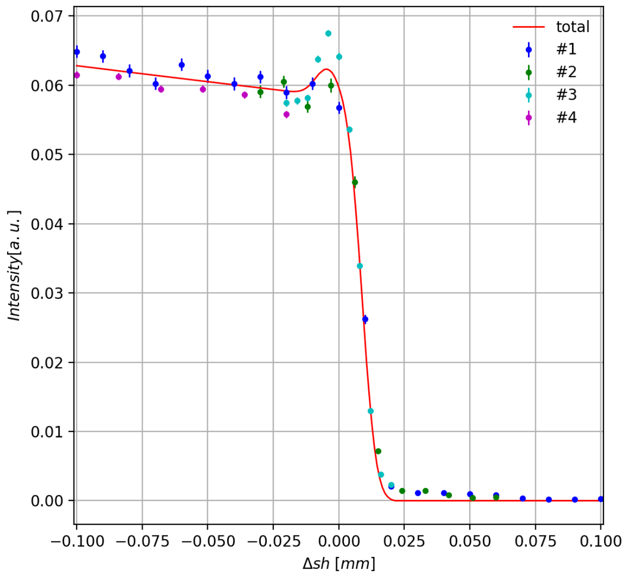
X: Sh

✓	top_scale	7.63e-09	
	bot_scale	5.53e-09	
	top_con.	1.00e-03	M
	sur_den.	5.00e-03	$Å^{-2}$
	bot_con.	0.00e+00	M
	bg	0.00e+00	
	curvature	200.0000	m
	Sh offset	5e-03	mm
✓	Qz offset	-4.80e-04	$Å^{-1}$
	Qz	0.0060	$Å^{-1}$

Simu range: -0.1:0.1 ✓ Show Components

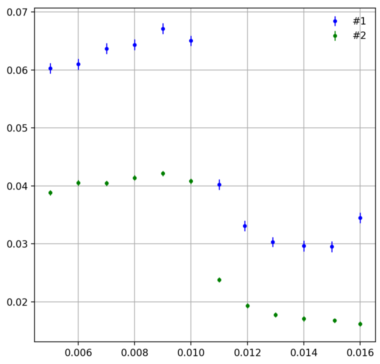
Fit Range: -0.1:0.1 Weight: data file

```
    # function evals   = 14
    # data points      = 11
    # variables        = 2
    chi-square         = 42.5533447
    reduced chi-square = 4.72814941
    Akaike info crit   = 18.8814950
    Bayesian info crit = 19.6772856
[[Variables]]
    losc:  5.53e-09 (fixed)
    hisc:  7.6256e-09 +/- 1.1039e-10 (1.45%) (init = 6.79e-09)
```

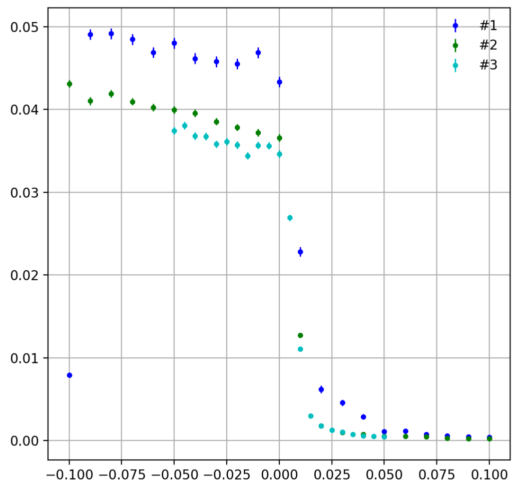


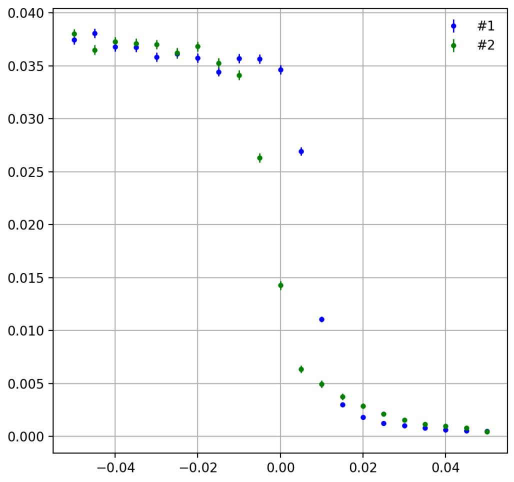
Sample 7. 1 mM Eu, 10 mM HDEHP / 0.95mM citric acid, 100 µM DTPA pH=3 (no adjustment)

Qscan: scan number: 585 (Green). The data is significant lower than #648 from sample 6 (blue) below critical angle even if there is commensurate amount of Eu ions in organic phase.



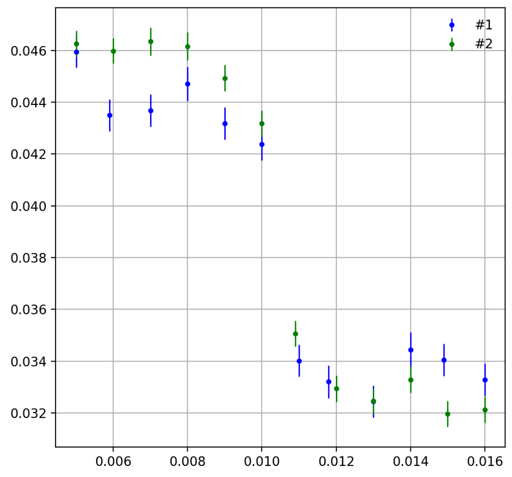
Shscan: left $Q_z = 0.006$ Å^{-1}, #687&688 (green & cyan), compare with #640 sample 6 (blue); right $Q_z = 0.015$ Å^{-1} #691 compared with #688(blue). Data is lower than the sample 6 data to a less extent. The sample height san at $Q_z = 0.015$ Å^{-1} explains why the data is significantly lower than pure water sample: there is a big sh offset.





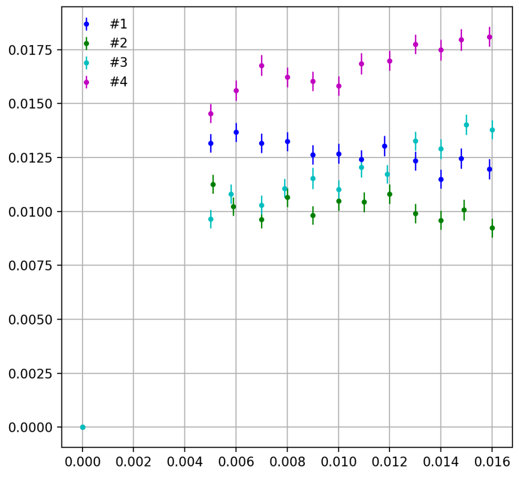
Sample 8. 1 mM Eu, 10 mM* HDEHP */0.55 mM nitric acid, 0.1 mM* DTPA, pH=*3 (no adjustment)

Qscan #736 compared with #595 from sample 4. Sample 8 is the repetition of the sample 4. Data is quite reproducible.

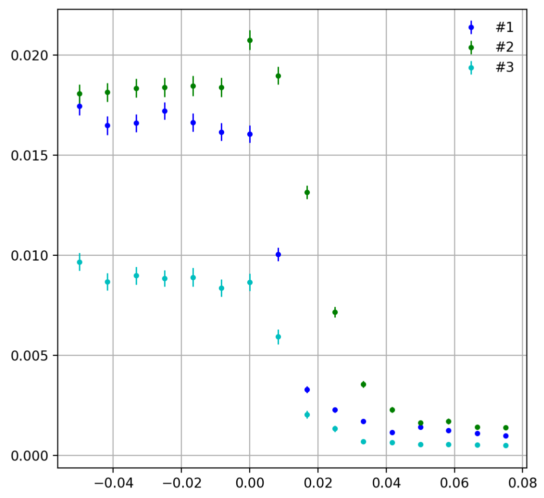


Sample 9. 1 mM Eu 10 mM* HDEHP */ 0.5M Citric 0.1mM* DTPA pH=*3 (adjusted with* NH_4OH*)

Qscan: #760, #761, #767, stir 20min, #772. This sample is not equilibrated when taking fluorescence data. Sample 10 was then measured to confirm the equilibrium.

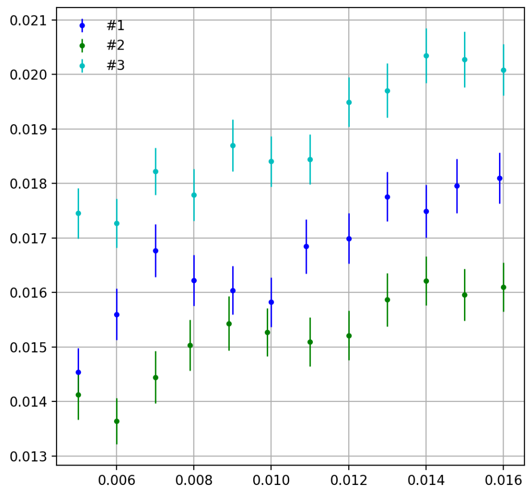


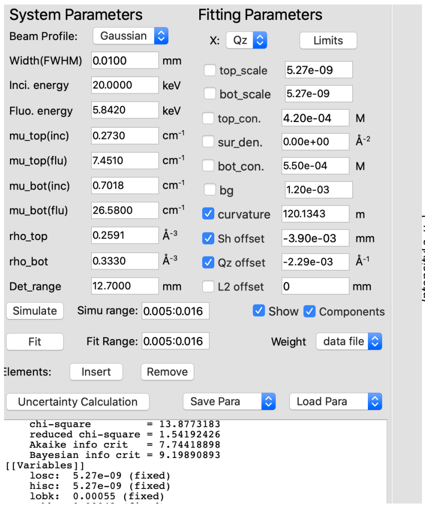
Shscan $Q_z = 0.006$ Å^{-1}. Scan number: #765 (cyan); #774 (blue) was taken after stirring for 20min, then #776 (green, $Q_z = 0.015$ Å^{-1}) was taken.



Sample 10. 1 mM Eu, 10 mM* HDEHP */ 0.5 M Citric acid, 0.1 mM* DTPA, pH=*3 (adjusted with* NH_4OH*)

Qscan: #802 (green) was taken first. After stirring for 10min, #804 (cyan) was taken, in comparison with #772 from sample 9 which was not equilibrated yet.





System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0100	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	Å^{-3}
rho_bot	0.3330	Å^{-3}
Det_range	12.7000	mm

Fitting Parameters

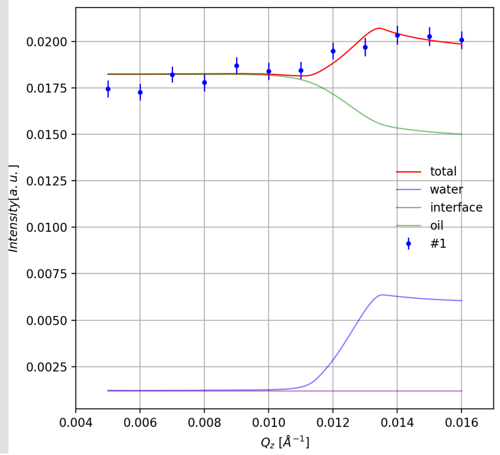
X: Qz

☐ top_scale	5.27e-09	
☐ bot_scale	5.27e-09	
☐ top_con.	4.20e-04	M
☐ sur_den.	0.00e+00	Å^{-2}
☐ bot_con.	5.50e-04	M
☐ bg	1.20e-03	
☑ curvature	120.1343	m
☑ Sh offset	-3.90e-03	mm
☑ Qz offset	-2.29e-03	Å^{-1}
☐ L2 offset	0	mm

Simu range: 0.005:0.016 ☑ Show ☑ Components

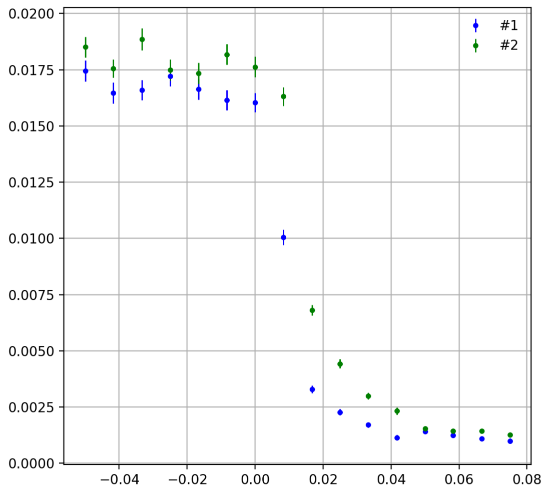
Fit Range: 0.005:0.016 Weight: data file

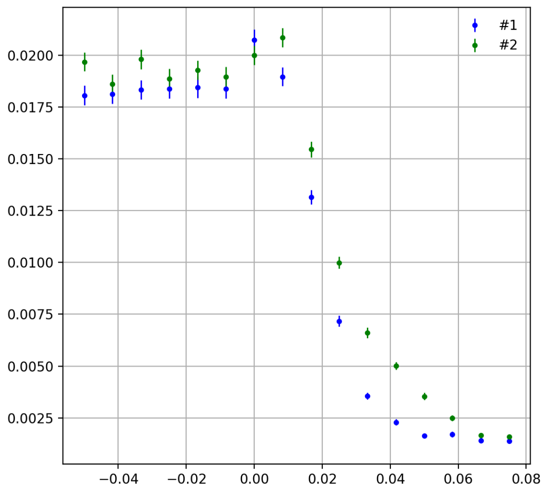
```
    chi-square          = 13.8773183
    reduced chi-square = 1.54192426
    Akaike info crit    = 7.74418898
    Bayesian info crit = 9.19890893
[[Variables]]
    losc:   5.27e-09 (fixed)
    hisc:   5.27e-09 (fixed)
    lobk:   0.00055 (fixed)
```

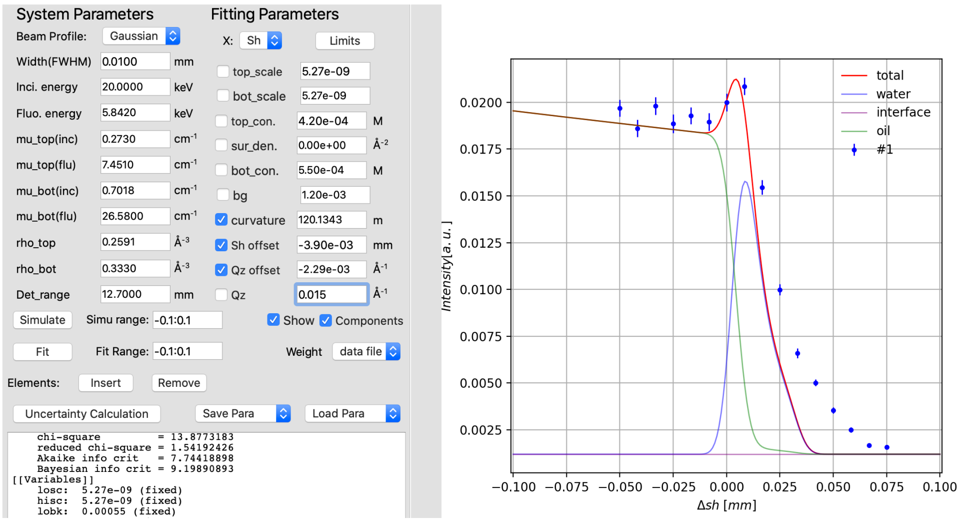


Shscan: $Q_z = 0.006$ Å^{-1} #806 (green) compared with #774 (blue from sample 9).

$Q_z = 0.015$ Å^{-1}: #807 (green) compared with #776 (blue) from sample 9. The intensity in the aqueous phase region (~0.04 mm) is higher than sample 9, indicating more Eu was extracted into aqueous phase.







System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0100	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	Å^{-3}
rho_bot	0.3330	Å^{-3}
Det_range	12.7000	mm

Fitting Parameters

X: Sh

☐ top_scale	5.27e-09	
☐ bot_scale	5.27e-09	
☐ top_con.	4.20e-04	M
☐ sur_den.	0.00e+00	Å^{-2}
☐ bot_con.	5.50e-04	M
☐ bg	1.20e-03	
✓ curvature	120.1343	m
✓ Sh offset	-3.90e-03	mm
✓ Qz offset	-2.29e-03	Å^{-1}
☐ Qz	0.015	Å^{-1}

Simu range: -0.1:0.1 ✓ Show ✓ Components

Fit Range: -0.1:0.1 Weight: data file

```
    chi-square         = 13.8773183
    reduced chi-square = 1.54192426
    Akaike info crit   = 7.74418898
    Bayesian info crit = 9.19890893
[[Variables]]
    losc:  5.27e-09 (fixed)
    hisc:  5.27e-09 (fixed)
    lobk:  0.00055 (fixed)
```

D.2 <u>Other important data for [HEH]EHP sample</u>

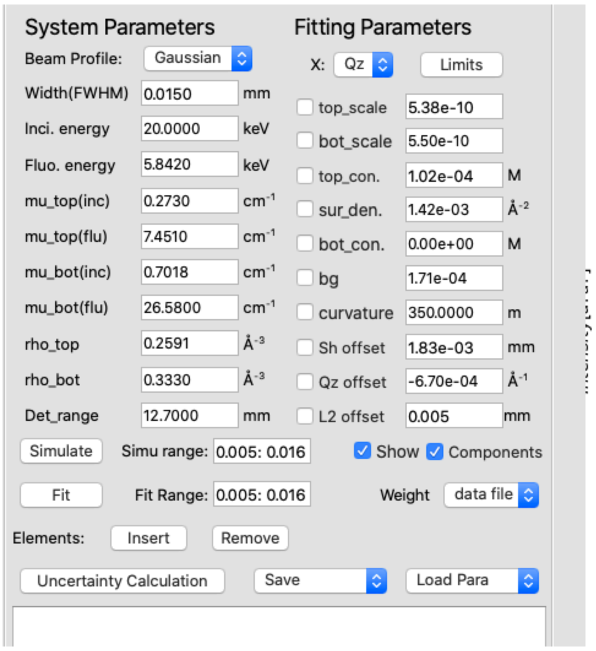
D.2.1 <u>2018 November run</u>

A wider beam was used for this run with “s1h 1 1”. FWHM is 15 µm.

The concentration of Eu in dodecane is limited to be 0.1 mM to avoid precipitates. As a result, the fluorescence intensity measured from the citric sample has very large error bars. More data needs to be taken to further demonstrate the fluorescence.

Sample 5 pure water / 5.8 mM HEHEHP, 0.1 mM Eu

It requires the existence of Eu on the interface, with the surface density 1.42(+0.20/-0.22)E-3 Å^{-2}. It is the first HEH[EHP] sample that provides evidence for the existence of excess Eu ions at the interface. Scan number: #368.



System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0150	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm⁻¹
mu_top(flu)	7.4510	cm⁻¹
mu_bot(inc)	0.7018	cm⁻¹
mu_bot(flu)	26.5800	cm⁻¹
rho_top	0.2591	Å⁻³
rho_bot	0.3330	Å⁻³
Det_range	12.7000	mm

Fitting Parameters

X: Qz

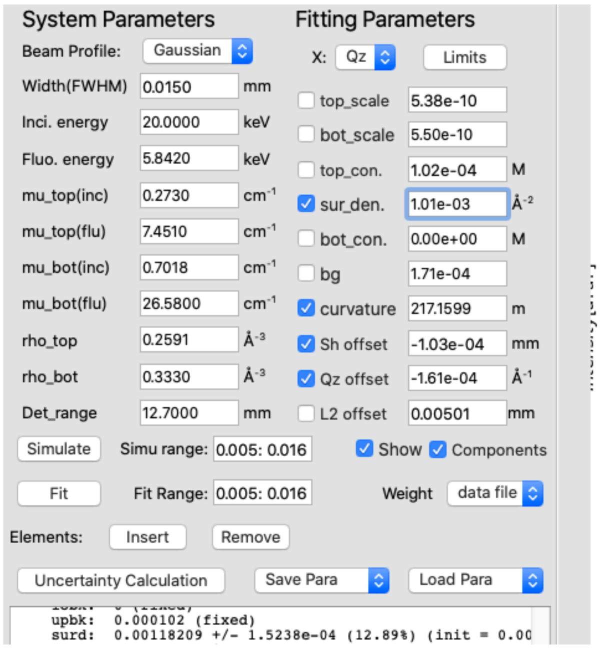
☐ top_scale	5.38e-10	
☐ bot_scale	5.50e-10	
☐ top_con.	1.02e-04	M
☐ sur_den.	1.42e-03	Å⁻²
☐ bot_con.	0.00e+00	M
☐ bg	1.71e-04	
☐ curvature	350.0000	m
☐ Sh offset	1.83e-03	mm
☐ Qz offset	-6.70e-04	Å⁻¹
☐ L2 offset	0.005	mm

Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight: data file

Sample 6 1mM HNO3 pH=3 / 5.8mM HEHEHP 0.1mM E

This sample indicates the excess interfacial ion in the presence of HNO_3. Scan number: 447,448,449 combined.



System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0150	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm⁻¹
mu_top(flu)	7.4510	cm⁻¹
mu_bot(inc)	0.7018	cm⁻¹
mu_bot(flu)	26.5800	cm⁻¹
rho_top	0.2591	Å⁻³
rho_bot	0.3330	Å⁻³
Det_range	12.7000	mm

Fitting Parameters

X: Qz

☐ top_scale	5.38e-10	
☐ bot_scale	5.50e-10	
☐ top_con.	1.02e-04	M
✓ sur_den.	1.01e-03	Å⁻²
☐ bot_con.	0.00e+00	M
☐ bg	1.71e-04	
✓ curvature	217.1599	m
✓ Sh offset	-1.03e-04	mm
✓ Qz offset	-1.61e-04	Å⁻¹
☐ L2 offset	0.00501	mm

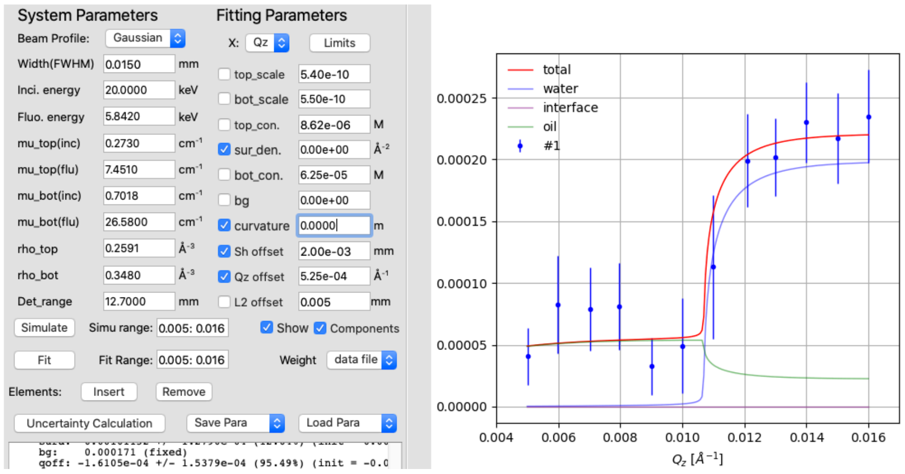
Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight: data file

```
     upbk:  0.000102 (fixed)
     surd:  0.00118209 +/- 1.5238e-04 (12.89%) (init = 0.00
```

Sample 7 0.5M citric pH=3 / 5.8mM HEHEHP

The pH value of the sample was adjusted with NH_4OH. The simplest model to fit the data was to have ions in both bulk phases but not at the interface. Citric acid extracts Eu ions back into water phase.



System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0150	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	$Å^{-3}$
rho_bot	0.3480	$Å^{-3}$
Det_range	12.7000	mm

Fitting Parameters

X: Qz

☐ top_scale	5.40e-10	
☐ bot_scale	5.50e-10	
☐ top_con.	8.62e-06	M
✓ sur_den.	0.00e+00	$Å^{-2}$
☐ bot_con.	6.25e-05	M
☐ bg	0.00e+00	
✓ curvature	0.0000	m
✓ Sh offset	2.00e-03	mm
✓ Qz offset	5.25e-04	$Å^{-1}$
☐ L2 offset	0.005	mm

Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight: data file

```
bg:    0.000171 (fixed)
qoff: -1.6105e-04 +/- 1.5379e-04 (95.49%) (init = -0.0
```

Critical Angle: Water/dodecane: 0.0103 $Å^{-1}$; 0.5M Citric/dodecane: 0.0112 $Å^{-1}$

Elemental analysis result

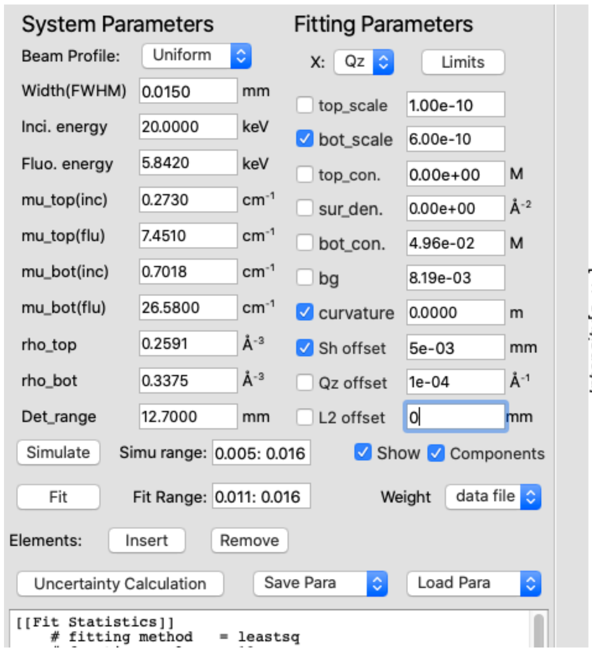
		Eu in aqueous (µg/L, mM)	Eu in dodecane (mM)
1	water/1mM Eu w/ 10mM HDEHP	5, 3.29E-5	1.089
2	Citrate/1mM Eu w/ 10mM HDEHP	9.4E5, 6.19	0.71
3	Nitrate/1mM Eu w/ 10mM HDEHP	5.85, 3.85E-5	1.02
4	0.05mM Eu/dodecane	6.64E6, 43.69	0
5	water/0.1mM Eu w/ 5.8mM HEHEHP	N/A	N/A
6	Nitrate/0.1mM Eu w/ 5.8mM HEHEHP	44.1, 2.90E-4	0.121
7	Citrate/0.1mM Eu w/ 5.8mM HEHEHP	1.07E5, 0.704	0.077

Atomic mass for Eu 152g/M. 1µg/L of Eu is 6.58E-6mM.

D.2.2 <u>2019 April run</u>

HEHEHP data from this run confirms the result obtained from Nov2018 run. But more data with higher statistics on the citric sample is still needed.

***Sample 6 dodecane / 50mM* Eu(NO_3)$_3$**



System Parameters

Beam Profile:	Uniform	
Width(FWHM)	0.0150	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	$Å^{-3}$
rho_bot	0.3375	$Å^{-3}$
Det_range	12.7000	mm

Fitting Parameters

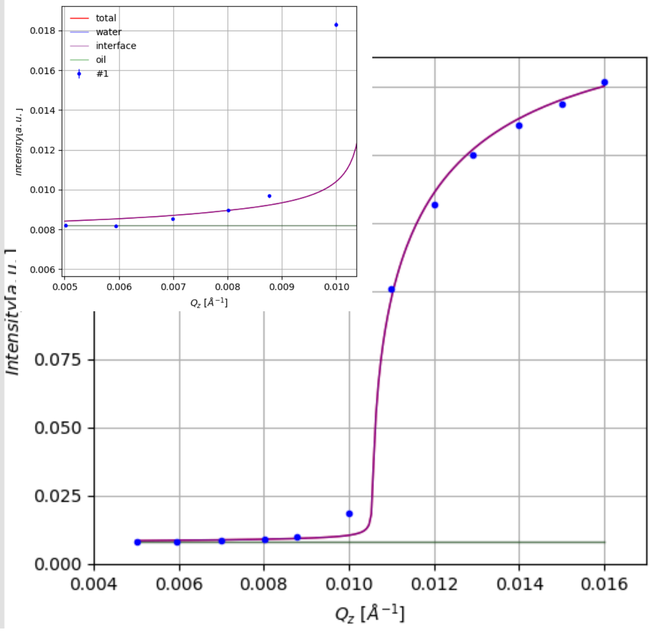
X: Qz

☐ top_scale	1.00e-10	
✓ bot_scale	6.00e-10	
☐ top_con.	0.00e+00	M
☐ sur_den.	0.00e+00	$Å^{-2}$
☐ bot_con.	4.96e-02	M
☐ bg	8.19e-03	
✓ curvature	0.0000	m
✓ Sh offset	5e-03	mm
☐ Qz offset	1e-04	$Å^{-1}$
☐ L2 offset	0	mm

Simu range: 0.005: 0.016 ✓ Show ✓ Components

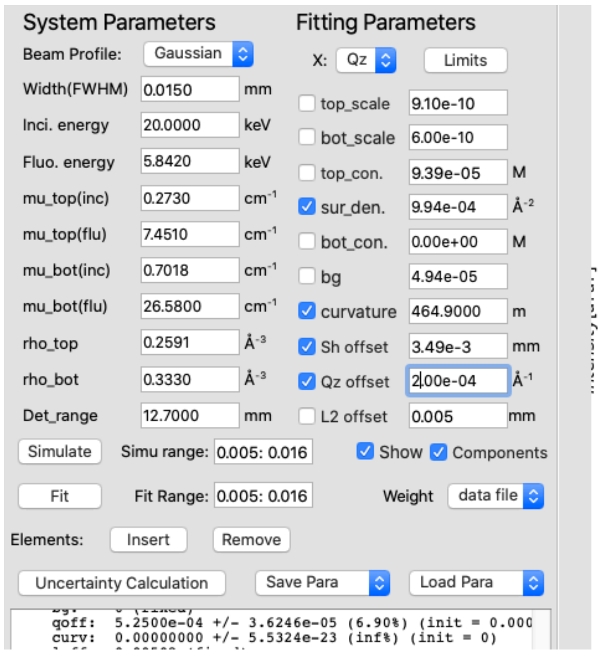
Fit Range: 0.011: 0.016 Weight: data file

```
[[Fit Statistics]]
    # fitting method    = leastsq
```



Sample 9 & 13 0.1mM Eu / 5.8mM HEHEHP/ pure water

Two samples with pure water: blue data is sample 9 with scan number 72 & 82 combined; green is sample 13 with scan number 431 & 432 combined. The red line fits to sample 13 data.



System Parameters

Beam Profile:	Gaussian	
Width(FWHM)	0.0150	mm
Inci. energy	20.0000	keV
Fluo. energy	5.8420	keV
mu_top(inc)	0.2730	cm^{-1}
mu_top(flu)	7.4510	cm^{-1}
mu_bot(inc)	0.7018	cm^{-1}
mu_bot(flu)	26.5800	cm^{-1}
rho_top	0.2591	$Å^{-3}$
rho_bot	0.3330	$Å^{-3}$
Det_range	12.7000	mm

Fitting Parameters

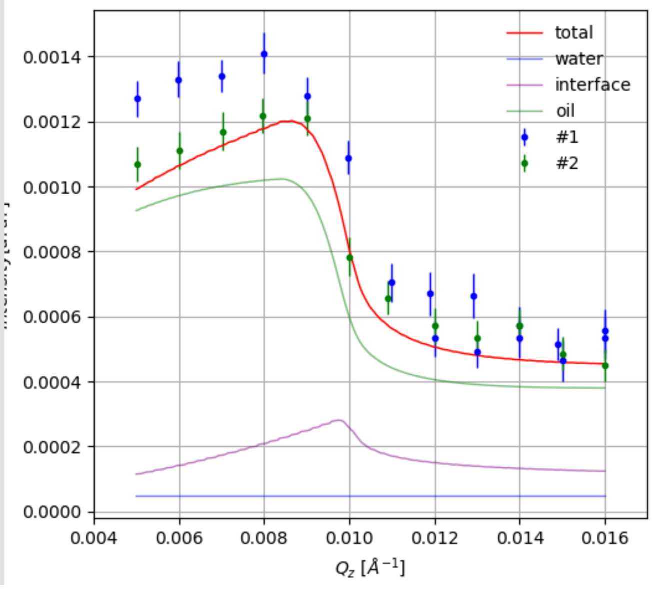
X: Qz

☐ top_scale	9.10e-10	
☐ bot_scale	6.00e-10	
☐ top_con.	9.39e-05	M
✓ sur_den.	9.94e-04	$Å^{-2}$
☐ bot_con.	0.00e+00	M
☐ bg	4.94e-05	
✓ curvature	464.9000	m
✓ Sh offset	3.49e-3	mm
✓ Qz offset	2.00e-04	$Å^{-1}$
☐ L2 offset	0.005	mm

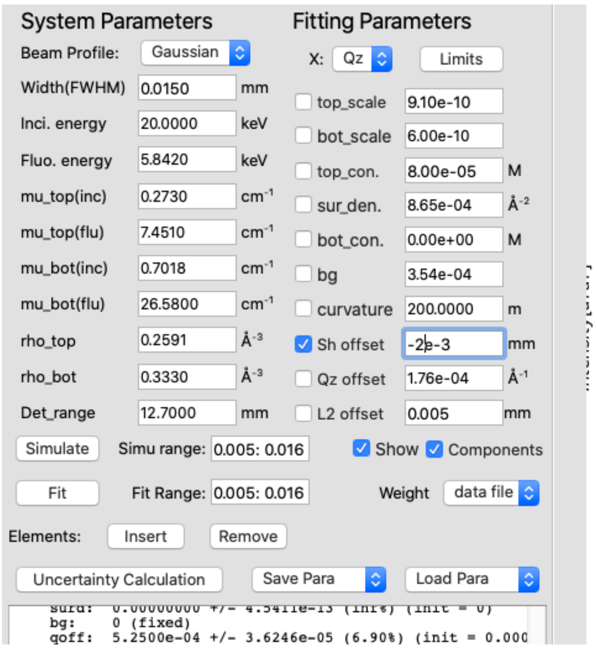
Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight: data file

```
    qoff:  5.2500e-04 +/- 3.6246e-05 (6.90%) (init = 0.000
    curv:  0.00000000 +/- 5.5324e-23 (inf%) (init = 0)
```



Sample 10 0.1mM Eu /HEHEHP 1mM HNO3



System Parameters

Beam Profile: Gaussian

Width(FWHM) 0.0150 mm

Inci. energy 20.0000 keV

Fluo. energy 5.8420 keV

mu_top(inc) 0.2730 cm^{-1}

mu_top(flu) 7.4510 cm^{-1}

mu_bot(inc) 0.7018 cm^{-1}

mu_bot(flu) 26.5800 cm^{-1}

rho_top 0.2591 $Å^{-3}$

rho_bot 0.3330 $Å^{-3}$

Det_range 12.7000 mm

Fitting Parameters

X: Qz

top_scale 9.10e-10

bot_scale 6.00e-10

top_con. 8.00e-05 M

sur_den. 8.65e-04 $Å^{-2}$

bot_con. 0.00e+00 M

bg 3.54e-04

curvature 200.0000 m

✓ Sh offset -2e-3 mm

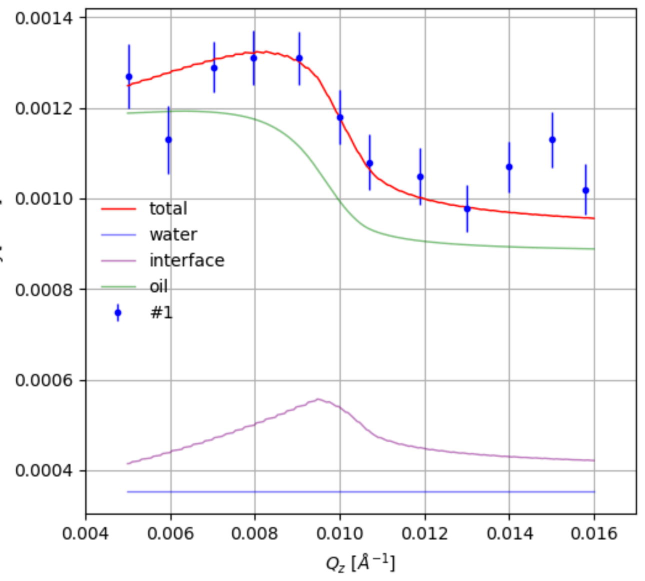
Qz offset 1.76e-04 $Å^{-1}$

L2 offset 0.005 mm

Simu range: 0.005: 0.016 ✓ Show ✓ Components

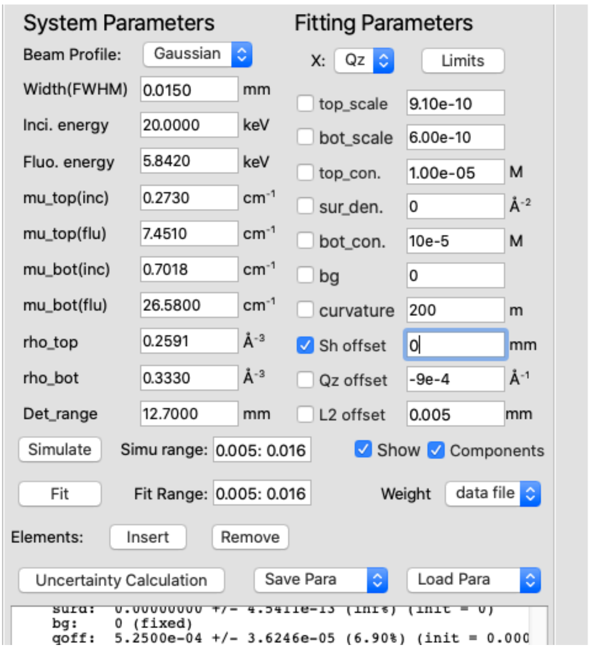
Fit Range: 0.005: 0.016 Weight data file

```
surd:  0.00000000 +/- 4.5411e-13 (inf%) (init = 0)
bg:    0 (fixed)
qoff:  5.2500e-04 +/- 3.6246e-05 (6.90%) (init = 0.000
```



Sample 11 0.1* mM Eu/HEHEHP, *0.5* M *Citrate* pH=*3 (adjusted with* NH_4OH*)

Scan 294, 299 and 306 merged.



System Parameters

Beam Profile: Gaussian

Width(FWHM) 0.0150 mm

Inci. energy 20.0000 keV

Fluo. energy 5.8420 keV

mu_top(inc) 0.2730 cm^{-1}

mu_top(flu) 7.4510 cm^{-1}

mu_bot(inc) 0.7018 cm^{-1}

mu_bot(flu) 26.5800 cm^{-1}

rho_top 0.2591 $Å^{-3}$

rho_bot 0.3330 $Å^{-3}$

Det_range 12.7000 mm

Fitting Parameters

X: Qz

top_scale 9.10e-10

bot_scale 6.00e-10

top_con. 1.00e-05 M

sur_den. 0 $Å^{-2}$

bot_con. 10e-5 M

bg 0

curvature 200 m

✓ Sh offset 0 mm

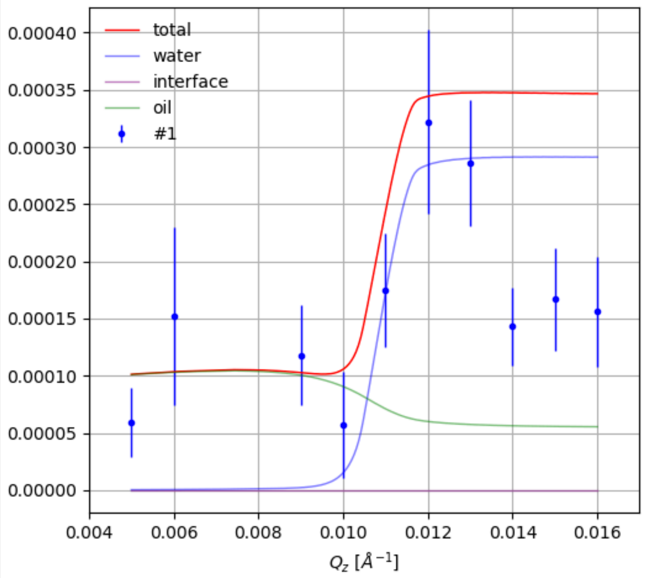
Qz offset -9e-4 $Å^{-1}$

L2 offset 0.005 mm

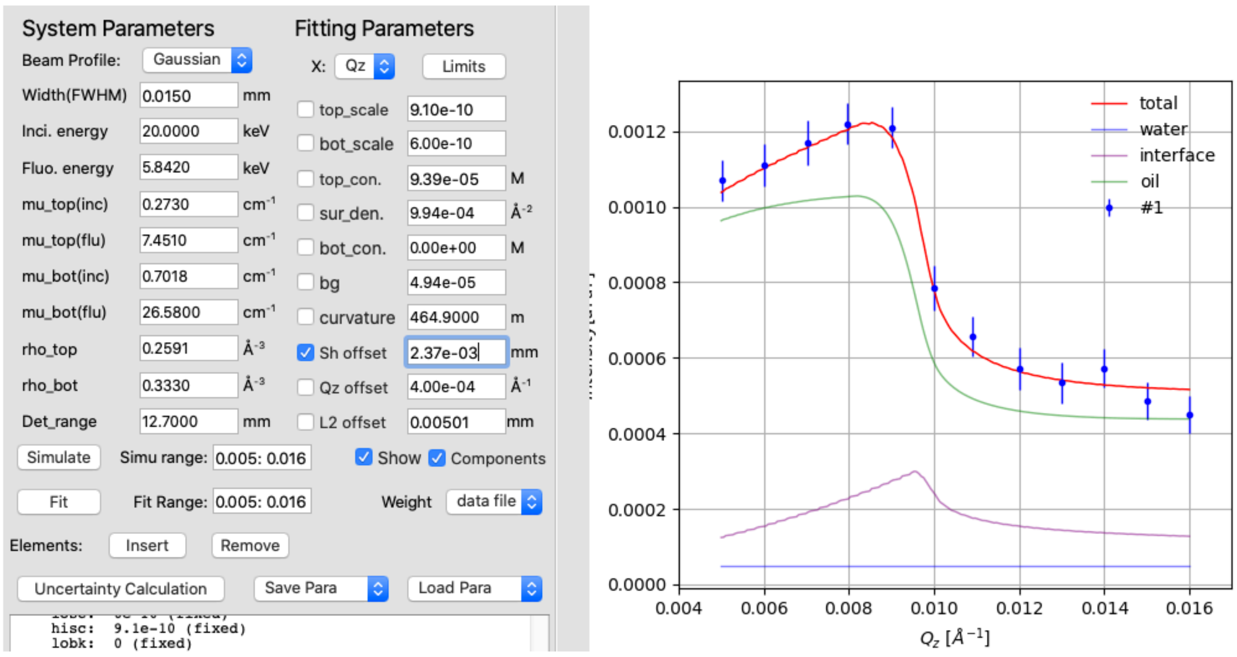
Simu range: 0.005: 0.016 ✓ Show ✓ Components

Fit Range: 0.005: 0.016 Weight data file

```
surd:  0.00000000 +/- 4.5411e-13 (inf%) (init = 0)
bg:    0 (fixed)
qoff:  5.2500e-04 +/- 3.6246e-05 (6.90%) (init = 0.000
```



Sample 13 0.1mM Eu 5.8 HEHEHP pure water



System Parameters

Beam Profile: Gaussian

Width(FWHM) 0.0150 mm

Inci. energy 20.0000 keV

Fluo. energy 5.8420 keV

mu_top(inc) 0.2730 cm^{-1}

mu_top(flu) 7.4510 cm^{-1}

mu_bot(inc) 0.7018 cm^{-1}

mu_bot(flu) 26.5800 cm^{-1}

rho_top 0.2591 $Å^{-3}$

rho_bot 0.3330 $Å^{-3}$

Det_range 12.7000 mm

Fitting Parameters

X: Qz

- [ ] top_scale 9.10e-10
- [ ] bot_scale 6.00e-10
- [ ] top_con. 9.39e-05 M
- [ ] sur_den. 9.94e-04 $Å^{-2}$
- [ ] bot_con. 0.00e+00 M
- [ ] bg 4.94e-05
- [ ] curvature 464.9000 m
- [x] Sh offset 2.37e-03 mm
- [ ] Qz offset 4.00e-04 $Å^{-1}$
- [ ] L2 offset 0.00501 mm

Simu range: 0.005: 0.016

[✓ Show] [✓ Components]

Fit Range: 0.005: 0.016

Weight data file

```
    hisc:  9.1e-10 (fixed)
    lobk:  0 (fixed)
```