Surface Ligand Binding Motif Chemistry on Semiconductor Nanoclusters and Quantum Dots

Casey Hannah Bisted

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Committee:

Munira Khalil, Chair

Brandi Cossairt

Stefan Stoll

Program Authorized to Offer Degree:
Chemistry

University of Washington

Abstract

Surface Ligand Binding Motif Chemistry on Semiconductor Nanoclusters and Quantum Dots

Casey Hannah Bisted

Chair of the Supervisory Committee:
Munira Khalil
Department of Chemistry

Semiconductor nanocrystals (NCs) offer utility across a range of optoelectronic devices and applications such as diodes, superlattice solar cells, and lasing. This versatility is due to their synthetically based size tunability, narrow linewidths, and strong absorptive cross sections. It is essential to understand the photophysical properties of semiconductor NCs such that they can be utilized to their fullest extent. Much of the photophysics of nanocrystals results from their core structure, but the surface defects and unbalanced charges of the core can be catastrophically destructive to these properties.

Organic ligands with carboxylic acid head groups can be used to stabilize these defects, and the interconversion of head group binding motifs in the ligand shell is dynamic with increased free ligand concentration, elevated temperature, and changes in solvent identity. Carboxylic acid head groups bind to the NC core surface with a variety of different binding motifs with indicative frequencies that can be detected with infrared spectroscopy. However, these frequencies are impacted by both ligand and core identity. Fitting the infrared signature of the carboxylic acid binding motifs with gaussians allows for characterization of surface ligand populations. This thesis documents the surface ligand binding populations and solution interaction of long

chain carboxylic acid ligands on InP magic size clusters and CdS Quantum Dots (QDs).

Chapter 1 describes an overview and perspective of carboxylic acid binding motifs and experimental methods used to characterize them. Emphasis is put on using infrared spectroscopy to identify binding motif and applied Bayesian inference to deconvolute complicated infrared spectra.

Chapter 2 documents the relative binding energies of carboxylic acid binding motifs on the surface of nanocrystals found through temperature dependent FTIR fit with a Markov-Chain Monte Carlo algorithm. The relative binding motif energies found through globally fitting a temperature dependent FTIR spectra of InP magic sized clusters (MSCs) with a Markov-Chain Monte Carlo global fitting algorithm are suggest previously unquantified binding motif lability. The chelating and syn-syn binding conformations are 0.7±0.3 kcal/mol and 1.1±0.5 kcal/mol more stable than the monodentate motif. Additionally, we find that the free ligand has nearly the same relative energy as the monodentate stretching frequency, with a relative energy of 4.52±0.05 kcal/mol, or 1582±19 cm^{-1}, which may suggest that ligand vibrational play a key role in ligand dissociation.

Chapter 3 documents how surface ligand populations change with the formation of macroscopic gels of CdS QDs. Through FTIR and ^{1}HNMR spectroscopy, we determine that the defused Z-type ligand cadmium oleate is essential for the formation of gels. We also determine that washing CdS QDs with ethanol successfully prevents gelation due to removal of weakly bound Z-type ligands from CdS surface. Finally, by comparing the IR spectra across temperature of these ethanol and acetonitrile washed QDs, we determine that there are two mechanisms for gelation – physical gelation and chemical gelation – that are spectroscopically distinct.

This research gives insight into the interface of semiconductor QD surface and

their ligands as well as how the surface ligands interact with solvent. This knowledge can be utilized for synthetic control of colloidal quantum dots to better optimize their physical properties. Likewise, QD ligand solvent interactions allow for optimization of QD nanostructures. Furthermore, characterization of the ground state allows us to begin quantification of ligand binding moiety populations and dynamics in the excited state.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEDICATION

To my mother, Jacqueline Levy, who always said I could.

GLOSSARY

: DFT Density Functional Theory

: DOSY Diffusion-Ordered Spectroscopy

: FTIR Fourier Transform Infrared Spectroscopy

: IR Infrared

: MCMC Markov-Chain Monte Carlo

: MSC Magic Sized Clusters

: NC Nanocluster

: NMR Nuclear Magnetic Resonance

: QD Quantum Dots

: UVVIS Ultraviolet Visible Spectroscopy

: XRD X-ray Diffraction

ACKNOWLEDGMENTS

This thesis or my graduate work would not have come to fruition without the support of my advisor, Professor Munira Khalil. My research is by no means her bread and butter or the bread and butter of the Khalil lab, but she gave me the freedom to explore, to question, and to go on data analysis "Rabbit Trails" as she calls them. She also gave me the freedom to fail, which is a luxury many don't have, and I failed many many times. Brandi Cossairt offered scientific advice , knowledge, and passion such that I would always look forward to our meetings. I am thankful for all the time she gave me, resources sent at weird morning hours, and pre-talk feed back I received . While I seldom met with Stefan, his care to know me as a person and understand my motivations as a scientist struck me. His ability to look at a scientific problem and deconstruct it is simultaneously fearsome and inspiring , and I appreciate the career advice he gave me .

From the Khalil Group, Megan Klein was integral to my learning from helping me trouble shoot my lasers together in the wee hours of morning to editing thesis chapters to helping me dive deep into data analysis and statistics. Moreso, she is a great friend and support . I am intensely grateful for her voice of reason and level head that got me through my third and fourth year. William Jeffries was also a great mentor with a non nonsense attitude to science. I appreciate his drive and holistic approach to problem solving. I never knew the Khalil Group without Caroline Loe. We met on our second day of grad school and became fast friends . Her attention to detail, integrity, and tenacity made me strive to be my best science wise and as a person. I would like to thank Amke Nimmrich for our coffee walks and sarcastic conversations. Likewise, I would like to thank Zhao Yuan Yang, Doyk Hwang, Ben

Poulter, William Miller , Izzy Sedwick, and Lily Von Felt for their camaraderie and moral support.

I would like to thank the UAW union for bargaining so hard each cycle such that I have great health insurance to get therapy and medication for my ADHD. Frankly, thank you who ever synthesized aderall first.

Outside of academia, I was supported first and foremost by my parents. My parents have always shown me that loving is doing. My parents who called me to make sure I was eating and taking care of myself when research and writing got rough. My parents who always gave a listening ear a phone call away. My mother who always knew I could and that I can, who imparted kindness with each word of advice. And, my father, with his steadfast wisdom of " You just get your certificate" and " Everyone wants to be a cowboy but no one likes to ride" .

My chosen family of Micaela and Caroline got me through my first several years of graduate school. I have never had friends that I could trust so wholeheartedly with my mental space. I appreciate the time we shared simply talking , complaining about the state of the world, and watching survivor and inkmaster. Even though pandemic sucked, it gave rise to the beauty of getting to know you two as people. I think wistfully of sitting in your guys' old apartment on 63rd calmly petting Violet and Lena.

I would like to thank my SH and ORG friends for the time spent together during pandemic and giving me social interaction where it was not possible otherwise. Thank you noot, for your clever insight and logic based thinking, Mufasa for calling me in lab when it was late my time and normal time hers, and AnT for reminding me there was always light at the end of the tunnel. I learned from my good friend Robby that while feedback may not sound kind it may be correct, which actually reframed a lot about how I viewed graduate school. Thank you Tena, Ashley, Monty, and Maximovic for sitting in discord calls while I gave practice talks. Last, I would like to thank Jeb. It is weird to realize that I met you on an awful online game and now I look forward

to our weekly chats. You are incredibly weird, but your perspective on Jobby gave me the first incentive to decouple my self worth from the science. In addition, your reminder of the acronym " IWFYMTSCMSFY" to me each time before I gave a stressful meeting will live rent free in my mind.

My housemate Kathleen offered me a soft landing pad and heartfelt support whenever I came home. Kathleen thank you for forcing me to watch awful christmas movies even though I am not one to enjoy Christmas, inundating me with the info from your multitudes of fandoms, and making up ludicrous scenarios about the world with me. (Poor corn) Our friendship has grown over the last few years of living together, and I am very happy about that. Also, thank you for our daughter, Sylvie, the demon unfortunately has knawed her way into my heart.

Thank you to my pre-UW friends, for reminding me always where I came from. To Quinne Hauth, my best friend. For giving me the context of real life always, and for the witty banter we have about books, fanfiction, and gossip. To Sam Reul, Sam Shephard, and Shannon Fender for inspiring me with their own scientific success and commiserating with me over data woes (of any medium really) as well as making time to call me and eat "with" me over the phone. To Kate Mason, Roy Cramer, and Eloise Fae who knew me in the before times where I was just a nerdy science white girl at chinese school.

And, finally thank you to Seattle Dodgeball for giving me a community and an outlet so different from that of academia. Where I felt my personality was cloistered in the confines of academia, I could truly be my self at dodgeball. Justin Pothoof and Douglas Ishii acted as great liaisons between worlds . I am thankful for Shota Barbeau, who always has had my back in all aspects of life . I could not asked for such a loyal , intelligent friend. I enjoy your vastly different perspective than mine , and I hope that someday I can learn your calmness. I am forever indebted to Erin Gabbard and Bill Fischer for allowing me to cowork with them when my lab mates were gone, without them this thesis would not have been completed. I am grateful for my team mates

on Whisper and My Female Brethren, specifically Missy Takahashi, Stef Schmuck, Taylor Green, Alex Graves,Elisa Westman, and Erin Greenlee for showing me that smart, driven women can be multifacetted and successful.

Chapter 1
INTRODUCTION

1.1 Motivations

To address the demands of the modern world, it is crucial to develop and understand materials for various applications. In recent years, nanomaterials have attracted significant interest due to their unique properties and high reactivity, which stem from their small size large surface area-to-volume ratio. Among these, colloidal quantum dots (QDs) — nanoscale crystals with semiconductor cores stabilized by surface ligands — have emerged as promising materials across a wide range of applications, from medicine to optics.[7,11,16] Their appeal lies in the ability to synthetically tune their size and shape, enabling precise control over their optical properties.[6,18] While the core structure and morphology of QDs play a critical role in determining their optical behavior, surface ligands are key to imparting chemical functionality and compatibility, thereby expanding their practical uses.[4,41]

Colloidal QD solutions have a broad spectrum of applications, and QDs can also be integrated into macro-structures, where their individual properties combine to yield collective functionalities.[24] In these complex assemblies, the identity of the surface ligands plays a crucial role in determining the overall integration and performance of the QDs within the superstructure.[44] Modifying these ligands can significantly enhance or alter the functionality of the resulting macro-structure, enabling tailored properties for specific applications.

Synthesizing QDs with precise physical sizes, morphologies, and specific ligand identities for targeted chemical applications is no small feat. A key aspect of optimizing this process is understanding how surface ligands connect to the QD core.

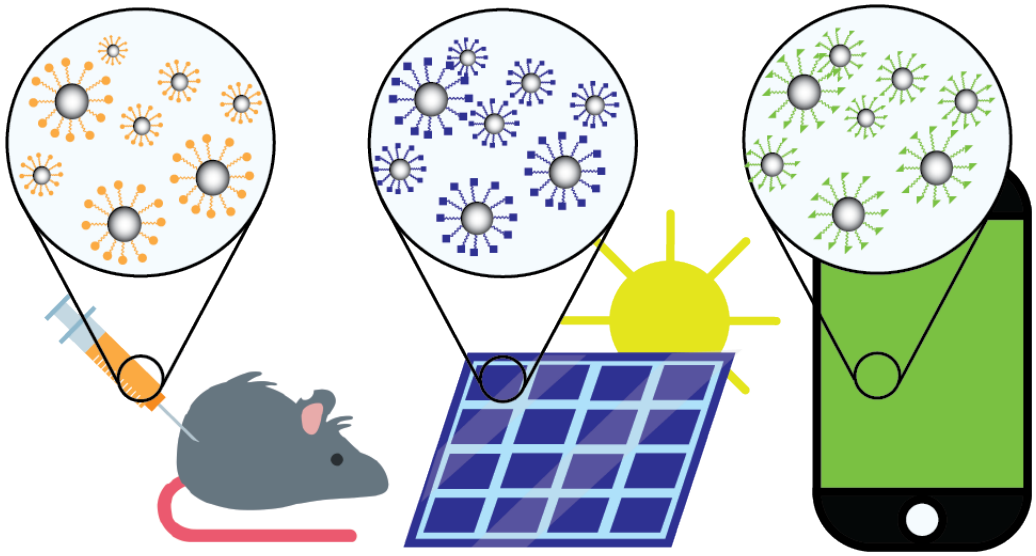

Figure 1.1: Various applications of QDs including medical imaging dyes, QDs used as photovoltaic material in solar cells, and QDs as light emitting diodes on a phone. Ligand identity is identified by different color and shape.

Ligands possess various head groups that chemically bond to the QD surface, and depending on identity, these head groups can adopt multiple binding conformations. As a result, the distribution of binding conformations can vary with ligand . Gaining insight into these binding populations is essential for the synthetic tailoring of QD surface chemistry as such enabling the customization of their chemical functionality for a wide range of applications.[2,45,47]

1.2 Semiconductor Nanocrystals Capped with Carboxylate Ligands

1.2.1 InP Magic Sized Clusters (MSCs)

InP QDs present a more environmentally friendly and non-toxic alternative to the widely studied Cd and Pb based QDs and offer promising applications in fields such as catalysis and LED displays.[21,25,26,37] However, achieving precise synthetic control over InP QDs remains a significant challenge.[38] Accurate characterization of surface bind-

ing conformations is essential not only for developing reproducible and size-tunable synthesis methods but also for understanding how to modify surface ligands post-synthesis.

InP QDs are typically synthesized through the heating of single-source precursors known as magic-sized clusters (MSCs), which are atomically precise nanoclusters stabilized by long-chain carboxylic acid ligands.[21,22,36] Due to the bidentate nature of carboxylic acids, these ligands can adopt multiple binding conformations on the InP surface.[46] Therefore, understanding the surface chemistry of these binding motifs on InP MSCs is beneficial in two ways: first, it provides an atomically precise model system for studying the more technologically relevant InP QDs, and second, it offers insights into the mechanism of InP QD growth from MSCs by revealing the distribution of binding motifs as well as their lability.

1.2.2 CdS QDs

CdS QDs possess a highly facile surface that is challenging to characterize.[5,19,39] Notably, CdS QDs are prone to forming gels during synthesis, with cadmium oleate impurities in CdS QD solutions believed to be responsible for this gelation.[40,44] This QD-embedded gel has potential applications in areas such as nanowire formation and gas adsorption.[17,24] However, the mechanism underlying its formation remains poorly understood. In many solution based gels, molecules key for gelation trap solvent between them thereby creating local molecular interactions and viscosity change.[10,20] Given that cadmium oleate is a precursor and surface ligand to CdS QDs, this poses the question of how do surface ligand binding motif impact ligand interaction with solvent. Investigating the surface ligand binding motifs on CdS QDs could provide valuable insights into the gelation process, thereby informing synthetic strategies to control gel formation and tune its properties for specific applications.

1.3 Methods for Carboxylate Ligand Binding Motif Identification

1.3.1 Carboxylate Binding Environments

Carboxylic acids can adopt several distinct binding modes on nanocrystal surfaces and be located in a variety of bonding environments. One such environment is the formation of Z-type complexes, where the carboxylate bonds stoichiometrically to the cation of the nanocrystal based on the cation's charge.[19,23] For instance, Z-type complexes on InP QDs typically take the form of In(carboxylate)$_3$, while on CdS QDs, they appear as Cd(carboxylate)$_2$.[36] These Z-type complexes stabilize the QDs by accepting electron pairs from surface anions.[15,21,29]

In addition to Z-type coordination, carboxylic acids can function as L-type ligands by donating electron pairs to the QD surface.[8,30] Lastly, carboxylates can bind covalently as X-type ligands, where they directly attach to the surface atoms, contributing to the QD's stability by neutralizing surface charges.[33,37] These ligand types can be seen in Figure 1.2.

X-type ligands can adopt multiple binding conformations, with several common motifs illustrated in Figure 1.2b.[33,46] While additional binding configurations do exist and combinations of these motifs can occur, the most frequently observed conformations are chelating, syn-anti bridging, syn-syn bridging, monodentate, and dative.[33] Prior to the research discussed in this thesis, bidentate interactions, such as chelating and bridging, were generally considered to be energetically favorable compared to monodentate bonding due to their increased denticity.

1.3.2 Ligand Binding Mode Identification

Characterizing surface ligand binding motifs on QDs is a complex challenge. Many existing techniques only provide information on a single QD rather than capturing the diversity of an entire sample, which may not accurately reflect the true population.[28,34] For example, X-ray diffraction (XRD) can reveal the crystal structure and associated

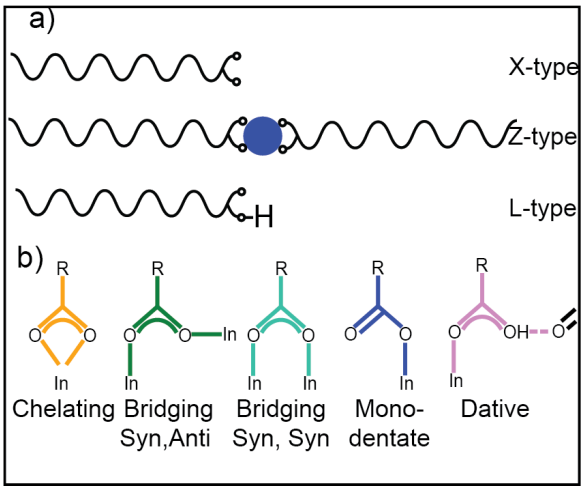

Figure 1.2: a) Ligand classification of a carboxylic acid into X-type, Z-type, and L-type. b) X-type carboxylic acid binding motifs arranged in order from lowest energy to highest energy in the asymetric stretching region

binding motif conformations of a nanocrystal as well as the binding environments of these ligands.[21,27,32] While XRD offers valuable insights into the identities of binding motifs present, it does not provide information on their interconversion or lability.

Furthermore, single-crystal X-ray diffraction is a solid-state technique, which may not accurately represent the surface binding populations of QDs in solution. Surface ligand dynamics, including the ability of binding motifs to interconvert, are likely to differ significantly between solid-state and solution-phase environments.[33] As such, understanding the full complexity of ligand binding on solution-based QDs requires alternative characterization techniques that can capture both the dynamic and heterogeneous nature of these surface interactions.

Nuclear Magnetic Resonance (NMR) is a solution-based characterization technique capable of differentiating between ligands that are bound to or free from the quantum dot (QD) surface, as well as providing insights into their local environments.[8,13] For instance, in ^{1}H NMR, the vinyl proton of oleic acid exhibits dis-

tinct peaks for bound and unbound species due to their differing electronic environments.[15,30] However, ^1H NMR alone cannot differentiate between free species such as Z-type metal complexes and L-type carboxylic acids or X-type binding motifs. To resolve this limitation of differentiating free species, Diffusion-Ordered Spectroscopy (DOSY) NMR is employed, as it leverages the differences in diffusion coefficients influenced by the size and shape of the analyte.[12,35] Given the significant size differences among QD-bound oleate, Z-type metal complexes, and L-type ligands, DOSY can effectively separate these species based on their diffusion rates, allowing for a more comprehensive analysis of their binding states.

The distinct binding motifs of X-type carboxylates exhibit unique vibrational spectral signatures that can be detected using Fourier Transform Infrared Spectroscopy (FTIR).[33,46] Each binding motif features characteristic asymmetric and symmetric carboxylate stretching vibrations. The frequency difference between these two stretches serves as a diagnostic tool for identifying ligand binding motifs, as it reflects variations in electron density within the carbon-oxygen bond.[14,46] In Figure 1.2, the binding motifs are arranged -according to the separation between their asymmetric and symmetric stretching frequencies, with chelating motifs showing the smallest difference and monodentate motifs displaying the largest. IR peaks associated with specific binding moieties can be fit with gaussians to compare population quantity and interconversion energies as can be seen in Figure 1.3.

1.4 Ligand Interconversion Energies through IR spectroscopy

Interconversion energies between different binding motifs can be analyzed using FTIR at variable temperatures. The Boltzmann equation describes the distribution of populations among different energy states as a function of temperature:

$$\frac{N_i}{N_j} = \alpha \exp\left(-\frac{\Delta E_{ij}}{k_{\mathrm{B}}T}\right) \tag{1.1}$$

where N_i and N_j are the populations of state i and state j, α is a proportionality

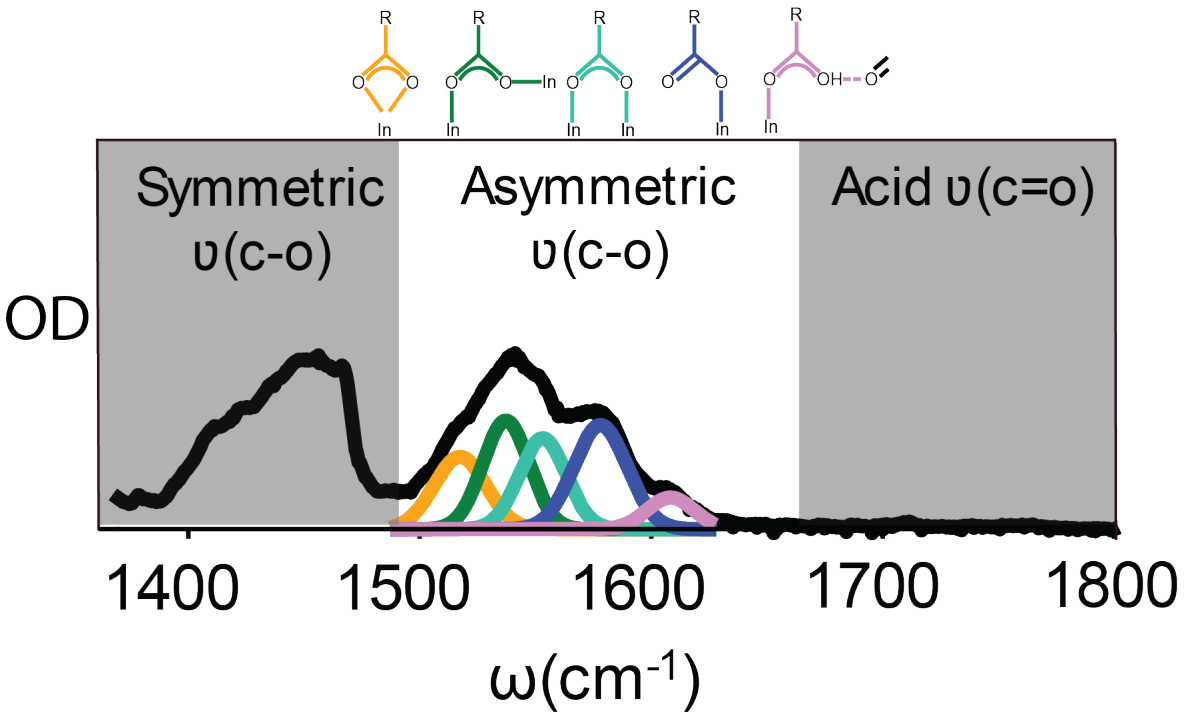

Figure 1.3: The FTIR spectra of myristic acid bound to InP Magic Sized clusters . The spectra , though convoluted , reveals 5 distinct binding motifs of the carboxylate head group in the assymetric C-O stretching region.

constant, ΔE_{ij} is the energy difference between the two states, k_B is the Boltzmann constant,T is the absolute temperature in Kelvin[9].[31]

At each temperature, the population percentages of all binding motifs are determined from the multi-gaussian fit of the FTIR spectra , allowing for a quantitative analysis via the Boltzmann equation to find their energy differences. This is demon-

strated in the solid state from Andreeva et al., where they utilized IR reflectance spectra to show how carboxylate ligands shifted from a tight to a loose state in metal organic frameworks.[1]

1.5 Utilizing Bayes' Theorem to find Posterior Distributions of Parameters

When modeling ligand populations, we often use multi-Gaussian equations to adequately model the complex lineshapes induced by multiple head group populations. An example of such convoluted spectra can be seen in Figure 1.3. However, it is crucial to assess how well the parameters of these equations represent the experimental data. This assessment is quantified by the posterior probability, denoted as $P(\theta|D, M)$, where θ represents the set of parameters of the model, D is the observed data, and M is the model function used for fitting.[42,43] More succinctly, the posterior probability is the probability density across the entire parameter space for a given model given observed data. A high posterior probability indicates that the model parameters are more likely to be accurate, with the spread of the posterior distribution reflecting the associated uncertainty.

The posterior probability is computed using Bayes' Theorem seen in Equation 1.2 where $P(D|\theta, M)$ is the likelihood, representing the probability of the data given specific parameters, $P(\theta|M)$ is the prior probability, reflecting our initial belief about the parameters before observing the data, and $P(D|M)$ is the evidence or the marginal, serving as a normalization constant to ensure that the posterior is a valid probability distribution.[3]

$$P(\theta|D, M) = \frac{P(D|\theta, M)P(\theta|M)}{P(D|M)} \tag{1.2}$$

By applying Bayes' Theorem, we can rigorously evaluate how well our model parameters fit the observed ligand population data, thus allowing for a robust analysis of the binding motif distributions. Posterior parameter spaces can be sampled through Markov-Chain Monte Carlo algorithms.

1.6 Thesis Outline

This chapter provides an overview of the capping ligands on semiconductor nanocrystals. Here I discuss two different semiconductor QD cores and their merit for study as well as how and why we should measure core ligand interactions on nanoclusters and quantum dots. Additionally, I discuss statistical methods for quantifying ligand binding species. Chapter 2 discusses the quantification and determination of ligand binding motif differences on InP Magic Size clusters via applied Bayesian statistics. Chapter 3 discusses how surface ligands interact with solvent molecules in CdS QDs in the formation of a macroscopic embedded QD Gels.

BIBLIOGRAPHY

(1) Andreeva, A. B.; Le, K. N.; Chen, L.; Kellman, M. E.; Hendon, C. H.; Brozek, C. K. *J. Am. Chem. Soc.* **2020**, *142*, 19291–19299.

(2) Baranov, A. V.; Ushakova, E. V.; Golubkov, V. V.; Litvin, A. P.; Parfenov, P. S.; Fedorov, A. V.; Berwick, K. *Langmuir* **2015**, *31*, 506–513.

(3) Bayes, R. M. *Philosophical Transactions of the Royal Society of London* **1763**, *53*, 370–418.

(4) Boles, M. A.; Ling, D.; Hyeon, T.; Talapin, D. V. *Nature Materials* **2016**, *15*, 141–153.

(5) Braam, D.; Mölleken, A.; Prinz, G. M.; Notthoff, C.; Geller, M.; Lorke, A. *Phys. Rev. B* **2013**, *88*, 1–6.

(6) Brown, P. R.; Kim, D.; Lunt, R. R.; Zhao, N.; Bawendi, M. G.; Grossman, J. C.; Bulović, V. *ACS Nano* **2014**, *8*, 5863–5872.

(7) Burda, C.; Chen, X.; Narayanan, R.; El-Sayed, M. A. *Chemical reviews* **2005**, *105*, 1025–1102.

(8) Cass, L. C.; Malicki, M.; Weiss, E. A. *Analytical Chemistry* **2013**, *85*, 6974–6979.

(9) Cercignani, C., *The Boltzmann Equation*; Springer: 1988.

(10) Chatterjee, S.; Kuppan, B.; Maitra, U. *Dalton Trans.* **2018**, *47*, 2522–2530.

(11) Cotta, M. A. *ACS Applied Nano Materials* **2020**, *3*, 4920–4924.

(12) Cros-Gagneux, A.; Delpech, F.; Nayral, C.; Cornejo, A.; Coppel, Y.; Chaudret, B. *J. Am. Chem. Soc.* **2010**, *132*, 18147–18157.

(13) De Roo, J.; De Keukeleere, K.; Hens, Z.; Van Driessche, I. *Dalton Transactions* **2016**, *45*, 13277–13283.

(14) Deacon, G. B.; Phillips, R. J. *Coord. Chem. Rev.* **1980**, *33*, 227–250.

(15) Drijvers, E.; De Roo, J.; Martins, J. C.; Infante, I.; Hens, Z. *Chemistry of Materials* **2018**, *30*, 1178–1186.

(16) Efros, A. L.; Rosen, M. *Annual Review of Materials Science* **2000**, *30*, 475–521.

(17) Enomoto, K.; Takeda, K.; Iwata, N.; Adachi, K.; Kikitsu, T.; Ishida, Y.; Hashizume, D.; Tanaka, M.; Kawakami, H.; Pu, Y.-J. *ACS Appl. Nano Mater.* **2022**, *5*, 3756–3762.

(18) Frederick, M. T.; Achtyl, J. L.; Knowles, K. E.; Weiss, E. A.; Geiger, F. M. *J. Am. Chem. Soc.* **2011**, *133*, 7476–7481.

(19) Fritzinger, B.; Capek, R. K.; Lambert, K.; Martins, J. C.; Hens, Z. *Journal of the American Chemical Society* **2010**, *132*, 10195–10201.

(20) Gaponik, N.; Herrmann, A.-K.; Eychmüller, A. *J. Phys. Chem. Lett.* **2012**, *3*, 8–17.

(21) Gary, D. C.; Flowers, S. E.; Kaminsky, W.; Petrone, A.; Li, X.; Cossairt, B. M. *Journal of the American Chemical Society* **2016**, *138*, 1510–1513.

(22) Ge, J.; Liang, J.; Chen, X.; Deng, Y.; Xiao, P.; Zhu, J.-J.; Wang, Y. *Chemical Science* **2022**, *13*, 11755–11763.

(23) Green, M. L. H. *Journal of Organometallic Chemistry* **1995**, *500*, 127–148.

(24) Hewa-Rahinduwage, C. C.; Geng, X.; Silva, K. L.; Niu, X.; Zhang, L.; Brock, S. L.; Luo, L. *J. Am. Chem. Soc.* **2020**, *142*, 12207–12215.

(25) Jang, E.; Kim, Y.; Won, Y.-H.; Jang, H.; Choi, S.-M. *ACS Energy Letters* **2020**, *5*, 1316–1327.

(26) Jang, E.; Jang, H. *Chemical Reviews* **2023**, *123*, 4663–4692.

(27) Jovanović, S.; Spreitzer, M.; Tramšek, M.; Trontelj, Z.; Suvorov, D. *J. Phys. Chem. C* **2014**, *118*, 13844–13856.

(28) Kamble, M. M.; Rondiya, S. R.; Bade, B. R.; Kore, K. B.; Nasane, M. P.; Dzade, N. Y.; Funde, A. M.; Jadkar, S. R. *Nanomater. Energy* **2020**, *9*, 72–81.

(29) Kirkwood, N.; Monchen, J. O. V.; Crisp, R. W.; Grimaldi, G.; Bergstein, H. A. C.; du Fossé, I.; van der Stam, W.; Infante, I.; Houtepen, A. J. *Journal of the American Chemical Society* **2018**, *140*, 15712–15723.

(30) Knauf, R. R.; Lennox, J. C.; Dempsey, J. L. *Chemistry of Materials* **2016**, *28*, 4762–4770.

(31) Lamm, G. *Reviews in Computational Chemistry* **2003**, *19*, 147–365.

(32) Lee, S. J.; Han, S. W.; Choi, H. J.; Kim, K. *J. Phys. Chem. B* **2002**, *106*, 2892–2900.

(33) Leger, J. D.; Friedfeld, M. R.; Beck, R. A.; Gaynor, J. D.; Petrone, A.; Li, X.; Cossairt, B. M.; Khalil, M. *Journal of Physical Chemistry Letters* **2019**, *10*, 1833–1839.

(34) Mohanan, J. L.; Arachchige, I. U.; Brock, S. L. *Science* **2005**, *307*, 397–400.

(35) Morris-Cohen, A. J.; Malicki, M.; Peterson, M. D.; Slavin, J. W. J.; Weiss, E. A. *Chem. Mater.* **2013**, *25*, 1155–1165.

(36) Ritchhart, A.; Cossairt, B. M. *Inorganic Chemistry* **2019**, *58*, 2840–2847.

(37) Ritchhart, A.; Cossairt, B. M. *Angew. Chem.* **2018**, *130*, 1926–1930.

(38) Ritchhart, A. Quantitative Analysis and Modification of Colloidal Nanoparticle Surfaces and Structures, Ph.D. Thesis, University of Washington, 2020.

(39) Shanavas, K. V.; Sharma, S. M.; Dasgupta, I.; Nag, A.; Hazarika, A.; Sarma, D. D. *J. Phys. Chem. C Nanomater. Interfaces* **2012**, *116*, 6507–6511.

(40) Shi, C.; Zhu, J. *Chem. Mater.* **2007**, *19*, 2392–2394.

(41) Smith, A. M.; Nie, S. *Acc. Chem. Res.* **2010**, *43*, 190–200.

(42) Speagle, J. S. *Phys ArXiv* **2019**, DOI: 10.48550/arXiv.1909.12313.

(43) Van Ravenzwaaij, D.; Cassey, P.; Brown, S. D. *Psychon. Bull. Rev.* **2018**, *25*, 143–154.

(44) Welsch, T. A.; Cleveland, J. M.; Thomas, J. A.; Schyns, Z. O. G.; Korley, L. T. J.; Doty, M. F. *ACS Appl. Nano Mater.* **2024**, *7*, 13319–13327.

(45) Xie, H.; Zeng, T.; Jin, S.; Li, Y.; Wang, X.; Sui, X.; Zhao, X. *Journal of Nanoscience and Nanotechnology* **2013**, *13*, 1461–1466.

(46) Zeleňák, V.; Vargová, Z.; Györyová, K. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy* **2007**, *66*, 262–272.

(47) Zhou, D.; Lin, M.; Liu, X.; Li, J.; Chen, Z.; Yao, D.; Sun, H.; Zhang, H.; Yang, B. *ACS Nano* **2013**, *7*, 2273–2283.

Chapter 2

MEASURING RELATIVE ENERGIES OF LIGAND BINDING CONFORMATIONS ON NANOCLUSTER SURFACES WITH TEMPERATURE-DEPENDENT FTIR SPECTROSCOPY

Reprinted with permission from Bisted et al..[36]

Klein, M. D.; Bisted, C. H.; Dou, F. Y.; Sandwisch, J. W.; Cossairt, B. M.;Khalil, M. J. Phys. Chem. C 2023, 127, 16970–16978.

We present a method to measure relative energies between binding conformations of carboxylate ligands on InP magic-sized clusters dissolved in solution. Using Markov Chain Monte Carlo global fitting analysis on temperature-dependent vibrational spectra of cluster-bound ligands, we observe significantly different relative energies between various bidentate and monodentate binding motifs. Relative to the monodentate motif, the chelating conformation is 0.7 ± 0.3 kcal/mol more stable, the syn-syn bridging conformation is 1.1 ± 0.5 kcal/mol more stable, but the syn-anti bridging conformation exhibits no significant difference. Our results find that the relative energy between monodentate-bound carboxylates and unbound carboxylic acids is 4.52 ± 0.05 kcal/mol, or 1582 ± 19 cm^{-1}, nearly identical to the carboxylate asymmetric stretching frequency. We suggest that the ligand vibrational energy may play a key role in ligand dissociation by compensating for energy differences between bound and dissociated ligand states. This approach gives important experimental insights into ligand binding and can inform future nanocrystal surface engineering.

2.1 Introduction

In recent years, semiconductor nanocrystal (NC) materials have demonstrated their utility across a range of optoelectronic applications. The emissive properties of these NCs have proven highly advantageous in the areas of light emitting diodes[31,69]and lasing,[41,49,56] while their absorptive behavior has led to their use in luminescent solar concentrators and numerous other photovoltaic and photocatalytic applications[11,13,38,50,67,70,73] . In particular, the size-tunability, narrow emission linewidths, strong absorption, and high quantum yield of nanomaterial platforms are desirable properties for numerous types of devices. Recent work has even begun investigating the potential for colloidally synthesized NCs to be utilized as photon sources in quantum information applications. In particular, the size-tunability, narrow emission linewidths, strong absorption, and high quantum yield of nanomaterial platforms are desirable properties for numerous types of devices. Recent work has even begun investigating the potential for colloidally synthesized NCs to be utilized as photon sources in quantum information applications.[12,53,65] In order to ensure these materials can be utilized to their fullest extent, it is crucially important to have a fundamental understanding of the properties that dictate NC photophysics.

While the core of a NC determines many of its characteristic photophysical properties, these properties are often highly sensitive to disruption by defects at the surface of the NC. It has been shown that undercoordinated surface anions are a predominant cause of sub-bandgap trap states that localize charges and dominate exciton recombination.[16,23,29,30,35,58,66,71] Due to this effect of surface anions, many NC syntheses are performed with an excess of cation precursors to ensure a cation-rich surface.[1,8] In addition, capping ligands are typically incorporated on the nanomaterial surface, playing a multifaceted role of passivating dangling surface bonds, ensuring charge balance, and promoting particle stability (both chemical and colloidal). Given the critical impact these ligands have on preserving NC properties, they are a subject of

significant and continued study in the field.[27,47,68] Numerous classes of both organic and inorganic ligands are used to functionalize nanocrystal surfaces. Organic ligands with long alkyl tails are a common example as they ensure surface passivation, prevent particle agglomeration, and enable NC suspension in organic solvents. These alkyl tails are frequently used in conjunction with carboxylate head groups, which act as X-type ligands: covalently bonding to surface cations, while also ensuring charge balance at the nonstoichiometric surface.[15,25,48] Despite the covalent ligand-NC bond, the ligand shell has been shown to be dynamic with frequent self-exchange aided by solvent, excess ligand, or elevated temperatures.[9,20,21]

To probe the dynamic nature and energetics of nanocrystal-bound ligands, previous studies have often relied on ligand exchange reactions. By monitoring the displacement of one ligand species following the addition of a second, information can be obtained on the difference in free energies between the types of ligands.[1,9,17,18,33,37,54,61]While these experiments are indisputably valuable in understanding binding differences between ligand species, they are unable to give insights into differences between binding motifs of a single species. In the case of carboxylate-capped InP nanocrystals, previous work by Gary et al. has shown that four different binding geometries can be observed via single crystal X-ray diffraction, with a fifth monodentate geometry observed in NMR studies.[22,54] There is much motivation to understand differences in binding motifs since they may also affect the surface reactivity and kinetics of nanocrystal growth, ultimately impacting factors such as shape and size dispersity for high-quality NCs.[76] However, unlike ligand displacement reactions, there is no easy handle by which to titrate between these binding motifs, making it very difficult to gain information about energetics between these configurations.

However, unlike ligand displacement reactions, there is no easy handle by which to titrate between these binding motifs, making it very difficult to gain information about energetics between these configurations.[51,52]However, it has been shown that with carboxylic acid ligands, the predominant binding species to II-VI and III-V semi-

conductor NCs is the aforementioned conjugate carboxylate anion.[10,14,21,54]However, it has been shown that with carboxylic acid ligands, the predominant binding species to II-VI and III-V semiconductor NCs is the aforementioned conjugate carboxylate anion.[5,46] While MX_2 ligands can be readily displaced via substitution with an L-type ligand,[1]in some systems such as InP magic-sized clusters (MSCs), it has been shown that metal ions are not displaced from the surface until well above 100 ^{o}C.[20]Despite this, the ligands are observed exchanging well below this temperature, meaning that the carboxylate species themselves are dynamically rearranging or dissociating from the surface[20]. Work by Xiet al. calculated the relative energies involved in rearrangement of carboxylate ligands in clusters of indium acetate complexes and found values ranging from 0-3 kcal/mol for converting between chelating and bridging conformations, and around 15-20 kcal/mol for converting between chelating and monodentate.[72]However, both of these values were strongly dependent on the surrounding environment of In atoms and ligand molecules and we would expect them to change drastically when going from an indium acetate cluster to the surface of an InP MSC. A study by Zhang et al. looked to calculate the binding energy of carboxylate moieties on facets of CdSe NCs and found a significantly larger difference of 50-80 kcal/mol between chelating and bridging conformations.[75] In the limit of low ligand density this relative energy difference decreased to 3 kcal/mol, which is much closer to the results calculated for the indium acetate clusters. While these computational results have generated some insight into the surface energetics of these binding motifs, there remains an urgent need for experimental measurements.

In this work, we present a temperature-dependent Fourier transform infrared (FTIR) spectroscopy method to study evolving populations of binding configurations for myristate-capped InP magic-sized clusters and extract relative energy differences between them. By utilizing the variation in the asymmetric COO- stretching frequencies between binding motifs, we globally fit the series of spectra to a set of five gaussian peaks across all temperatures via a Markov Chain Monte Carlo (MCMC) process. We

then used the changes in peak areas to find the relative energy differences between the binding configurations of carboxylate groups, as well as the free carboxylic acid species. From these results, we observe that the energetic separation between a monodentate ligand and a free ligand is approximately equal to the vibrational stretching frequency of the carboxylate group. This strongly suggests a connection between vibrational excitation of the ligand and its dissociation from the surface. Additionally, we observe that the difference between bidentate and monodentate can be far smaller - and for some bidentate configurations the additional bond may not provide any increased stability at all. This experimental approach enables measuring energy differences between binding motifs, offers unique insight into NC surface energetics, and further suggests that the ligand shell may be even more dynamic than previously thought.

2.2 Methods

2.2.1 InP MSC Synthesis and Characterization

Indium phosphide magic-sized clusters were prepared following the previously reported procedure [22]. A 100 mL 3-neck round bottom flask was charged with 0.936 g indium acetate (3.2 mmol, 1 eq) and 2.66 g myristic acid (11.5 mmol, 3.6 eq), and heated at 105 °C overnight with stirring under vacuum. The resulting indium myristate was then dissolved in 20 mL anhydrous toluene. In a nitrogen-filled glovebox, 500 μL of $P(SiMe_3)_3$ (1.72 mmol, 0.54 eq) was added to 10 mL anhydrous toluene, and the solution was transferred to a syringe. The contents were rapidly injected into the reaction flask. The reaction then proceeded at 105 °C until no further changes were observed by UV-Vis (Cary Agilent 500), approximately 50 minutes. The MSCs were then cooled, and the toluene was removed by vacuum distillation. The clusters were brought into the glovebox, redissolved in toluene, and precipitated with acetonitrile. This procedure was repeated for a minimum of three rounds of precipitation. A final

spin was done in pentane alone as the solvent to remove any undissolved species. The clusters were purified by size exclusion chromatography over 4 Åmolecular sieves in toluene and stored dry as a waxy solid.

2.2.2 FTIR Measurements

FTIR measurements were conducted in a Jasco 4100 FTIR spectrophotometer under a positive pressure of N_2. An 8 mg/mL solution of MSCs was flowed through a liquid cell with a 300 μ m Teflon spacer. Heated measurements were conducted with a temperature controlled demountable liquid cell (Harrick) paired with a Peltier heater at 1 °C/min. The sample was left to equilibrate for 10 minutes at each temperature point. Flowing was stopped when recording spectra and then resumed to ensure thermal equilibrium.

2.2.3 Markov Chain Monte Carlo Sampling

A homebuilt sampling algorithm was constructed in Python 3.8 following the parallelizable walker update scheme laid out by Foreman-Mackey et al.[19] Affine invariant sampling was used in the form of the stretch move described by Goodman and Weare.[24] A prior distribution was constructed based off a room temperature fit of the InP MSC FTIR spectrum (Figure A.3), with standard deviations equal to those of the fit parameters. Following a proposed step, the predicted spectrum was generated by least squares fitting the amplitude of the five gaussian peaks, corresponding to the proposed point in parameter space, to the FTIR spectrum at each temperature point. The log likelihood was then calculated as the negative sum of squared residuals. In order to ensure the assumption of normally distributed error was met, the fits and data were converted to transmission prior to calculating the residuals. Chains were thinned down to every 10 steps, and runs were typically performed with 200 walkers and a total of 300,000 proposed moves. The acceptance proportion was kept between 0.2 and 0.4 by adjusting the step size of the stretch move prior to a run if needed. The

correlation plot for the parameters is shown in Fig. A.4. The elliptical distributions of all of the peak parameter correlations demonstrate adequate parameterization of the model. Additionally, the results with the given prior distribution were validated by performing the MCMC global fitting again using wider prior distributions, as well as once more with uninformed priors.The results for these runs are shown in Figs. A.4 - A.6 and compared in Tables A.1 and A.2. Importantly, while we observe an increase in the uncertainty moving to wider and then uninformed prior distributions, we do not see a significant change in the extracted parameters or relative energies.

2.3 Results and Discussion

To study the energetics of carboxylate ligand binding modes on NC surfaces, we used myristate-capped InP magic-sized clusters, which have a precisely known formula and structure allowing for more rigorous analysis than larger and more heterogeneous nanoparticles. The physical structure of the phenylacetate variant of these MSCs was characterized via single crystal X-ray diffraction and the electronic structure was probed using UV-Vis spectroscopy (Fig. A.1).While the MSC consists of a nearly stoichiometric $[In_{21}P_{20}]^{3+}$ core, the surface is cation rich and exhibits a dense and interconnected indium carboxylate ligand network,[22] resulting in a molecular formula of $In_{37}P_{20}(O_2CR)_{51}$ where, in this case, R represents the myristate alkyl tail, $C_{13}H_{27}$.[6,54] Because 51 X-type carboxylate ligands are stoichiometrically bound to the InP MSC core for charge stabilization[20,54] , surface ligand density cannot be synthetically tuned without changing the chemical identity of the MSC. However, in the presence of free carboxylic acid, L-type adsorption of carboxylic acids on MSCs accounts for up to 15% of the ligated species[22,54] - impacting surface saturation.

Figure 2.1 plots the temperature-dependent FTIR spectra of the carboxylate vibrations of the bound (1400–1600 cm^{-1}) and free (1700–1780 cm^{-1}) myristate ligands. The bidentate nature of the carboxylate head group gives rise to five surface ligand binding motifs—chelating, syn-anti bridging, syn-syn bridging, monodentate, and

dative—illustrated in Figure 2.1. Each binding conformation results in a symmetric (s) and asymmetric (as) carboxylate stretching vibration. The CH_2 bending mode from the myristic acid alkyl tail overlaps the symmetric stretching region and in this work, we will focus on the asymmetric carboxylate stretching region. We use ab initio DFT frequency calculations of carboxylate stretching vibrations on the MSC to assign frequencies to the five binding motifs in the asymmetric carboxylate stretching region.

Our previous work showed that the chelating binding motif is always the lowest energy vibrational frequency in the asymmetric region between 1500 cm^{-1} -1510 cm^{-1}.[42] Similarly, the monodentate binding motif was computationally determined to be one of the highest frequency binding motifs, with peak centers experimentally determined to be around 1580 cm^{-1}. Of the two bridging motifs, the asymmetric stretching mode of syn-syn conformation is higher in frequency than that of syn-anti due to the electron repulsion of the two syn bonds in the same plane.[74] Thus, in the asymmetric region the frequency order will be: chelating ¡ syn-anti ¡ syn-syn ¡ monodentate ¡ ionic/dative.

It is worth noting that, in addition to bound carboxylate peaks from 1400 cm^{-1} to 1600 cm^{-1}, the free ligand peaks at 1710 cm^{-1} and 1760 cm^{-1} observed in Fig. 2.1 can also offer insight on surface ligand equilibrium. Specifically, the free carboxylic acid dimer arises at 1710 cm^{-1} with the monomer at 1760 cm^{-1}.[59,64] We confirm in Fig. A.3 of the SI that these peaks are due to desorbed and protonated carboxylic acid species through an equilibrium comparison of the monomeric and dimeric peak areas.

To characterize the bound carboxylate features, we fit the asymmetric stretching region of the FTIR spectrum at 25 °C to five gaussians associated from lowest to highest energy with the chelating, syn-anti bridging, syn-syn bridging, monodentate, and dative, as shown in Fig. 2.1. While at lower temperatures, there is sufficient resolution of the peaks for the spectra to be fit to five gaussians, at higher temperatures, peak broadening increases uncertainty of typical fitting algorithms and is unable to accurately fit the spectrally congested regions of the spectrum as reported by the

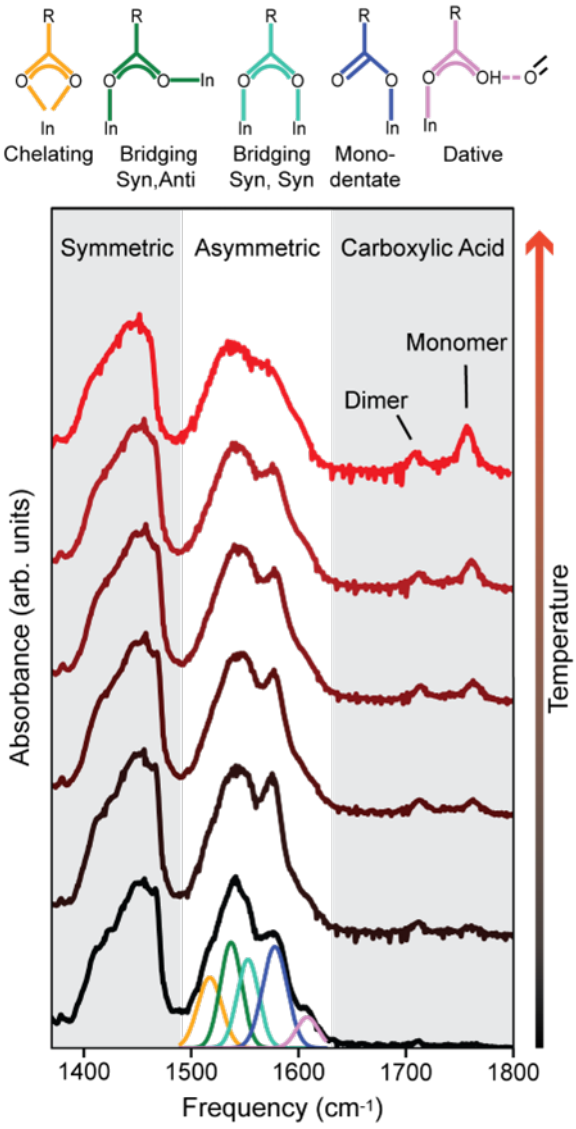

Figure 2.1: . Representative FTIR spectra from the temperature-dependent measurement of myristate-capped InP MSCs spanning from 25 °C (bottom) to 100 °C (top) in increments of 15 °C and demonstrating changes that occur with increasing temperature along with peak assignment of asymmetric carboxylate stretching region. From left to right, binding groups shown include chelating (yellow), bridging syn-anti (green), bridging syn-syn (light blue), monodentate (dark blue), and dative (pink).

high variance due to poor fitting. This broadening could be explained in part by the work of Friedfield et al. that observed a strong endothermic change in a differential scanning calorimetry (DSC) thermogram at 115-130 °C, which they attributed to crystal melting.[20] As the crystal nears the melting temperature at the high end of our temperature range, we would expect higher phonon occupancy and increased geometric disorder that would lead to vibrational linewidth broadening like we observe. As such, a more rigorous method is needed to perform a global fitting and analyze the full temperature FTIR manifold concurrently.

Extracting quantitative information from complicated spectra often necessitates peak fitting to determine the frequency, width, and amplitude of each component in the spectra. For a single spectrum consisting of only a few peaks, this is relatively straightforward to do with a simple non-linear least squares algorithm. In cases where a series of spectra are measured, and certain parameters are not expected to vary across the series, a global fit across the whole series can be performed at the cost of significantly increased complexity. To perform a global fit on the temperature-dependent FTIR spectra reported in this paper, we utilized a Bayesian approach with a MCMC algorithm to sample the parameter space and fit the spectra to a sum of five peaks across all temperatures. Briefly, the use of a Monte Carlo approach allows stochastic sampling of the parameter space, rather than relying on gradient-based methods which may get stuck in local minima.[2] Meanwhile, the use of a constructed Markov Chain satisfying ergodicity and detailed balance ensures the existence of a single stable solution. To quantify how likely a specific set of parameters is, we turn to Bayes' Theorem to calculate the posterior probability for a particular stochastically sampled fit (Eq. 2.1).

$$P(\theta \mid x) = \frac{P(\theta) \cdot P(x \mid \theta)}{\int P(\theta) \cdot P(x \mid \theta)\, d\theta} \tag{2.1}$$

Where the posterior probability, P(x), represents the probability that a set of parameters, θ, are correct given the observed data, x.[4] This is proportional to the product of the prior probability, P(θ), which describes the chance of a particular

set of parameters occurring using some prior knowledge of the system, multiplied with the likelihood, P(x—θ), which describes the probability of observing the data given a specific value for the parameters. This product is then normalized to give a probability distribution that integrates to one across the space. Because we are only concerned with finding the maximum posterior probability though, we only need to utilize relative changes in probability and can therefore neglect the computationally expensive integral.

To fit the FTIR spectra reported here, we constructed a model consisting of five Gaussian peaks, each with a mean frequency μ_i and a room-temperature width of σ_i. While we assume that the mean vibrational frequencies will not change significantly over the temperature range we probe, as mentioned above, some broadening is observed at elevated temperatures. The exact form of this broadening is too complex to readily model, as it would require a detailed knowledge of the phonon density of states as well as the nature of how the phonon modes affect the ligand binding geometry. Furthermore, trying to account for those behaviors in detail would result in a vastly overparameterized fit. As such, we rely on a simplified power law broadening term that captures the general behavior of the sum width, with each peak following the same broadening parameter, β. The σ_i parameters are treated as constants across temperatures and the width of peak i is then given by $\sigma_i \left(\frac{T}{T_0}\right)^{\beta}$, with T_0 taken to be 298 K. The full equation for the fit is therefore as follows in Equation 2.2, where $\tilde{\nu}$ represents the frequency and $c_i(T)$ is the temperature-dependent amplitude for peak i, as discussed in more detail below.

$$f(\tilde{\nu}, T) = \sum_{i=1}^{5} c_i(T) \exp\left(-\frac{(\tilde{\nu} - \mu_i)^2}{2\sigma_i^2 \left(\frac{T}{T_0}\right)^{2\beta}}\right) \tag{2.2}$$

With the basis of our fit established, we determined appropriate prior distributions through a non-linear least squares fit of the initial room temperature spectrum, as shown in Fig. A.4 . The MCMC scheme is depicted in Fig.2.2a . An ensemble

of Monte Carlo walkers is distributed in parameter space, and in each MCMC iteration a subset of the walkers is selected and random moves are generated to new proposed positions. The spectral peaks corresponding to these proposed parameters are then constructed, and the temperature-dependent amplitudes are determined by least squares fitting at each temperature point. The fit residuals are evaluated and used to estimate the likelihood function of the parameters. Based on the resulting posterior probability, the moves are then stochastically accepted or rejected and the ensemble positions are updated accordingly. The algorithm iterates through the remaining subsets of walkers and then repeats the process until a preset number of steps have been evaluated. The results of this global fitting process are shown in Fig. 2.2b, where the data is shown in orange and the average fits are shown by the blue lines with standard deviations at each point given by the blue region. Because we expect the best fit to occur at the maximum posterior probability, we performed statistical analyses on the set of 15,000 sampled points with the highest posterior probabilities. The obtained parameters and their associated uncertainties are shown in Table 2.1. We note that these values agree closely for a room temperature spectrum with those found previously for MSCs with oleate ligands, which possess a similar carboxylate binding group with a long alkyl tail.[42,64]

From the global fitting results, we can extract amplitude information without needing to enforce any assumptions on the temperature-dependent behavior of the underlying populations. This allows a more unbiased approach to see how binding motif populations change with temperature. In particular, we can look at the differences of the logs of peak areas to determine whether these populations follow a Boltzmann distribution at any point. This is shown in Fig. 2.3a. as the log area difference with respect to the monodentate binding mode peak, plotted against inverse thermal energy. Above 40 °C (to the left of the black dashed line in the figure) we observe a linear trend to the log area differences, suggesting that a Boltzmann distribution accurately describes the population, as shown in Equation 2.3. The change in

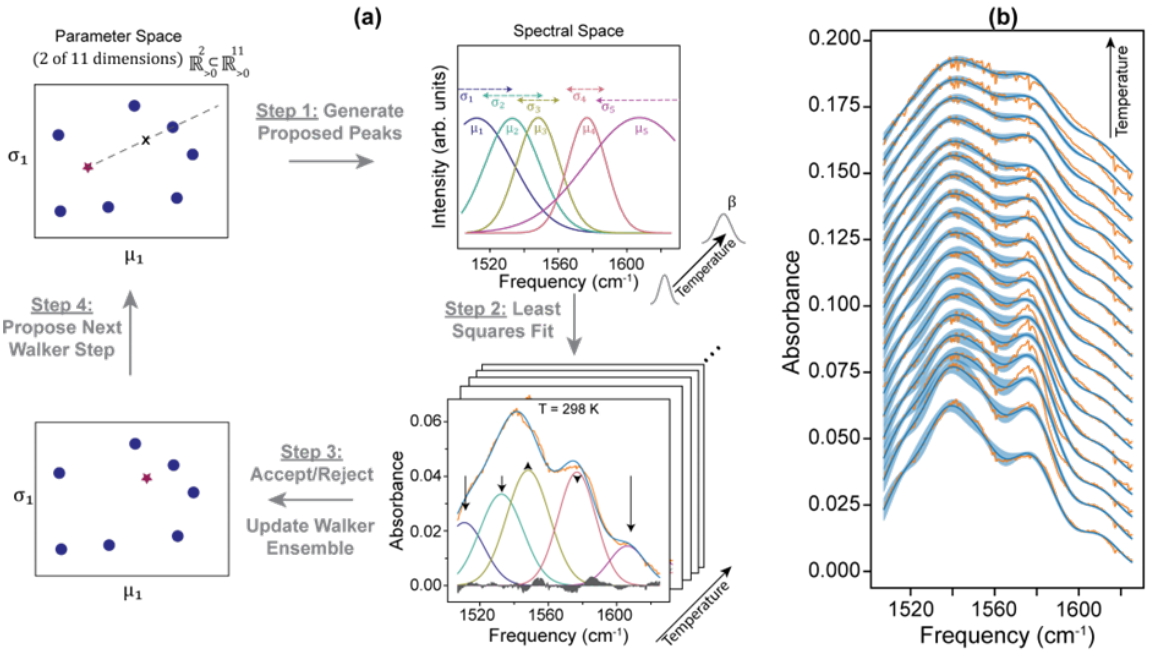

Figure 2.2: A) Depiction of the MCMC global fit process. For a given subset of walkers, stretch moves are randomly generated to propose new positions in parameter space. These new walker positions are used to generate the Gaussian peaks corresponding to those parameters with means μ_i and widths σ_i, along with a temperature-dependent broadening factor β. These peaks are then fit by least squares to the measured FTIR spectra at each temperature point. The log likelihood of this fit is taken as the negative sum of squared residuals; combined with the prior probabilities of the parameters, the resulting posterior probability is used to stochastically decide whether to accept or reject the new walker positions. The walker ensemble is then updated, and the algorithm iterates to the next subset of walkers and repeats the process. B) Global MCMC fit results (blue lines) for the temperature-dependent InP MSC FTIR spectra (orange lines). The standard deviation across the top 15,000 fits is depicted by the blue shaded area.

Table 2.1: Extracted parameters from the MCMC global fit on a temperature series of FTIR spectra from an InP MSC sample. Peaks were fit to a gaussian lineshape with position μ_i and width $\sigma_i(T/T_0)^\beta$, as shown in Equation 2. Values and errors are given as means and standard deviations among the sampled points in the top 15,000 posterior probabilities, respectively.

μ_1 (cm^{-1})	σ_1 (cm^{-1})	μ_2 (cm^{-1})	σ_2 (cm^{-1})	μ_3 (cm^{-1})	σ_3 (cm^{-1})
1515.0 ± 2.1	10.8 ± 1.0	1537.1 ± 1.3	11.2 ± 1.9	1553.9 ± 1.9	9.1 ± 0.8
μ_4 (cm^{-1})	σ_4 (cm^{-1})	μ_5 (cm^{-1})	σ_5 (cm^{-1})	β	
1577.0 ± 0.28	12.2 ± 0.27	1609.0 ± 0.28	10.0 ± 0.18	0.62 ± 0.10	

behavior at 40 °C suggests that once there is sufficient thermal energy, the system can fully equilibrate between various binding motifs and dissociated free ligand. This is in line with the previous DSC and ^{13}C NMR work by Friedfeld et al. on these clusters, who observed an endothermic event and linewidth broadening, respectively, occurring at slightly elevated temperatures that they ascribed to ligand rearrangement.[20]

We proceeded to perform linear regressions on the temperature-dependent log area differences in the high-temperature regime. Taking the data above 40 °C for the fits corresponding to the top 15,000 posterior probabilities sampled by the model, we found the averages and standard deviations of the slopes obtained from those regressions. These values are plotted in Fig. 2.3. Through application of the Boltzmann equation, we see that the resulting slopes correspond to the relative energy differences between these binding motifs with respect to the monodentate motif, shown by Equation 2.3.

$$\ln(A_{\text{mono}}) - \ln(A_i) = \ln\left(\alpha e^{\frac{\Delta\epsilon}{k_B T}}\right) = \ln(\alpha) + \frac{\Delta\epsilon}{k_B T} \tag{2.3}$$

Where A_i denotes the peak area of component i, $\Delta\epsilon = \epsilon_i - \epsilon_m ono$ is the change in

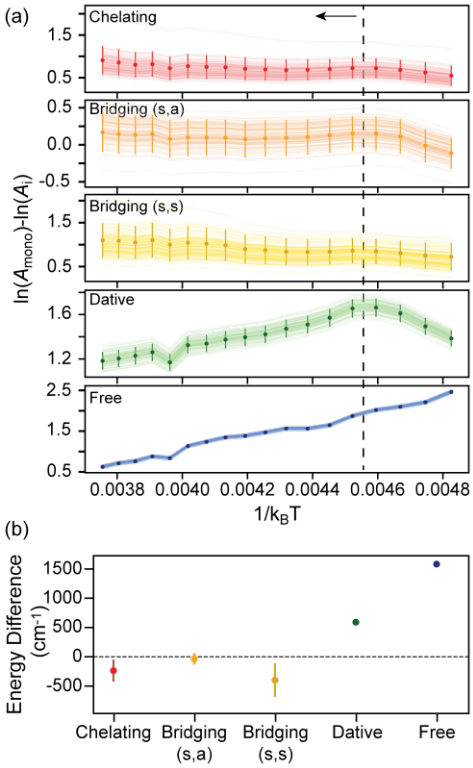

Figure 2.3: a) Differences of the log of peak areas with respect to the monodentate binding motif, plotted against inverse thermal energy with units of inverse wavenumbers. Markers and error bars represent mean and standard deviation at each temperature point across sampled global fits comprising the top 15,000 posterior probabilities. Lighter lines depict log area differences from the fits corresponding to the 100 highest posterior probabilities. Note the highly correlated error between temperature points which is not conveyed from the error bars alone. While log area differences might vary in magnitude between fits, the temperature trends stay rather consistent. Arrow denotes data points above 40 °C (black dashed line), which demonstrate near linear behavior and were fit using linear regression. b) Extracted slopes from linear regressions of the 45-110 °C regime in panel (a). These values correspond to energy differences relative to the monodentate binding motif, assuming an equilibrated system following Boltzmann statistics. Markers and error bars respectively denote mean and standard deviation of slopes taken from sampled global fits comprising the top 15,000 posterior probabilities.

energy between the two motifs, T is the temperature, k_B is the Boltzmann constant, and α is a proportionality constant.

From Fig. 2.3, we can clearly see a varied energy landscape between different binding conformations. While the chelating and syn-syn bridging conformations are lower in energy compared to the monodentate motif, the difference is not as large as might be expected from the formation of an additional ligand-metal bond. The energy difference we observe is on the order of 200-400 cm−1, or 0.57-1.14 kcal/mol, far lower than the 15-20 kcal/mol predicted by theory.[72,75] Notably, the syn-anti bridging conformation is not significantly different in energy from the monodentate motif, suggesting that the energetic bonus of an additional ligand-metal bond is offset by a confounding factor, potentially such as geometric strain.[28,62] Meanwhile, the dative binding conformation we observe has a higher energy relative to the monodentate. This dative motif is difficult to precisely assign to a geometry; previous single crystal studies show a single ligand in an asymmetric chelating geometry with a bridging dative bond, but these studies were done at -173 °C with phenylacetate ligands and it is unknown whether this preferred geometry will change to a monodentate, dative motif at higher temperatures or with a longer alkyl tail.[33] However, the vibrational energy we observe for this dative binding conformation agrees with general understanding on the increased stability of covalent X-type ligand bonds compared to dative L-type bonding on a cationic surface. Moreover, studies on ruthenium dimer complexes with carboxylic acid ligands have observed similar stretching frequencies for datively bound carboxylic acids that formed hydrogen bonds to neighboring carboxylates.[57] As L-type ligands do not contribute to charge balance on the surface, we would expect them to be able to convert or dissociate more readily like we see here. Finally, we observed a relative energy difference of 1582 ± 19 cm−1, around 4.5 kcal/mol, between the monodentate binding motif and the free ligand. This is of particular note, as this energy is not significantly different from the vibrational frequency,we extract for the monodentate asymmetric stretching mode, 1577.0 ± 0.28 cm−1. These results are

also summarized in Table 2.2.

Table 2.2: Summary of relative binding energies extracted from slopes of temperature-dependent log area differences.

Binding Motif	**Relative Energy (w.r.t. Monodentate)**
Chelating	-236 ± 120 cm^{-1}
Bridging (s,a)	-35 ± 57 cm^{-1}
Bridging (s,s)	-400 ± 184 cm^{-1}
Dative	591 ± 34 cm^{-1}
Free Ligand	1582 ± 19 cm^{-1}

While these energies do not provide information on the activation barriers between these conformations, they do inform on the relative stability of binding motifs compared to one another. These results suggest that the excitation of this vibrational stretching mode is enough to offset the energy loss associated with ligand dissociation. This is critically relevant for semiconductor nanocrystals where numerous processes, including thermalization to the band edge and Auger-Meitner recombination of trions or multiexcitons, result in energy loss as phonon emission.[3,26,32,43,44] This heat could easily dissipate through the ligand shell resulting in dissociation of the binding group and a decrease in photocatalytic or optical device performance. As we have shown, temperature-dependent FTIR spectroscopy provides the capability to directly investigate these evolving populations of ligand binding motifs. In particular, nanocluster platforms offer a well-defined structure and a strictly finite set of possible local surface environments, limiting heterogeneity that could impede interpretation of data obtained using this method. There are numerous other semiconductor clusters whose surface chemistry could be readily probed using this method, including

other III-V clusters,[39,40,55] and the broad range of II-VI clusters of distinct structure and stoichiometry[7,34,45,60,63] .While these measurements were performed in solution, nanocluster behavior often varies in the solid state. Understanding ligand behavior in device-relevant conditions would be highly advantageous to design principles, and this could be an area of further development to tailor this technique to the more disordered solid-state systems, such as larger nanocrystals.

2.4 Conclusion

In this work we have demonstrated that a temperature series of vibrational spectra can be used to measure relative energies of carboxylate binding conformations on the surface of InP magic-sized clusters. A Markov Chain Monte Carlo process can be used to optimize a global fit across all spectra and estimate fit uncertainty. We further show that the relative energy between these binding motifs is much lower than previously thought. A ligand with one bond to the surface is not significantly different in energy from a bidentate ligand bridging two In atoms in a syn-anti geometry. Lastly, we demonstrate that the energy difference between a monodentate binding motif and a free ligand is equal to the energy of the monodentate carboxylate asymmetric stretching mode. This suggests that the ligand-functionalized surface of a nanocrystal is far more dynamic and labile than previously thought, and that ligand vibrational excitation has the potential to impact surface binding coverage.

BIBLIOGRAPHY

(1) Anderson, N. C.; Hendricks, M. P.; Choi, J. J.; Owen, J. S. *Journal of the American Chemical Society* **2013**, *135*, 18536–18548.

(2) Ashner, M. N.; Winslow, S. W.; Swan, J. W.; Tisdale, W. A. *Journal of Physical Chemistry A* **2019**, *123*, 3893–3902.

(3) Auger, P. *Comptes Rendus de l'Académie des Sciences (C.R.A.S.)* **1923**, *177*, 169–171.

(4) Bayes, R. M. *Philosophical Transactions of the Royal Society of London* **1763**, *53*, 370–418.

(5) Bealing, C. R.; Baumgardner, W. J.; Choi, J. J.; Hanrath, T.; Hennig, R. G. *ACS Nano* **2012**, *6*, 2118–2127.

(6) Boles, M. A.; Ling, D.; Hyeon, T.; Talapin, D. V. *Nature Materials* **2016**, *15*, 141–153.

(7) Bootharaju, M. S.; Baek, W.; Lee, S.; Chang, H.; Kim, J.; Hyeon, T. *Small* **2021**, *17*, e2002067.

(8) Busby, E.; Anderson, N. C.; Owen, J. S.; Sfeir, M. Y. *Journal of Physical Chemistry C* **2015**, *119*, 27797–27803.

(9) Calvin, J. J.; O'Brien, E. A.; Sedlak, A. B.; Balan, A. D.; Alivisatos, A. P. *ACS Nano* **2021**, *15*, 1407–1420.

(10) Cass, L. C.; Malicki, M.; Weiss, E. A. *Analytical Chemistry* **2013**, *85*, 6974–6979.

(11) Chakraborty, I. N.; Roy, S.; Devatha, G.; Rao, A.; Pillai, P. P. *Chemistry of Materials* **2019**, *31*, 2258–2262.

(12) Chen, Y.; Sharp, D.; Saxena, A.; Nguyen, H.; Cossairt, B. M.; Majumdar, A. *Advanced Quantum Technologies* **2022**, *5*, 2100078.

(13) Coropceanu, I.; Bawendi, M. G. *Nano Letters* **2014**, *14*, 4097–4101.

(14) Cros-Gagneux, A.; Delpech, F.; Nayral, C.; Cornejo, A.; Coppel, Y.; Chaudret, B. *Journal of the American Chemical Society* **2010**, *132*, 18147–18157.

(15) De Roo, J.; De Keukeleere, K.; Hens, Z.; Van Driessche, I. *Dalton Transactions* **2016**, *45*, 13277–13283.

(16) Dolai, S.; Nimmala, P. R.; Mandal, M.; Muhoberac, B. B.; Dria, K.; Dass, A.; Sardar, R. *Chemistry of Materials* **2014**, *26*, 1278–1285.

(17) Drijvers, E.; De Roo, J.; Martins, J. C.; Infante, I.; Hens, Z. *Chemistry of Materials* **2018**, *30*, 1178–1186.

(18) Elimelech, O.; Aviv, O.; Oded, M.; Banin, U. *Nano Letters* **2020**, *20*, 6396–6403.

(19) Foreman-Mackey, D.; Hogg, D. W.; Lang, D.; Goodman, J. *Publications of the Astronomical Society of the Pacific* **2013**, *125*, 306.

(20) Friedfeld, M. R.; Johnson, D. A.; Cossairt, B. M. *Inorganic Chemistry* **2019**, *58*, 803–810.

(21) Fritzinger, B.; Capek, R. K.; Lambert, K.; Martins, J. C.; Hens, Z. *Journal of the American Chemical Society* **2010**, *132*, 10195–10201.

(22) Gary, D. C.; Flowers, S. E.; Kaminsky, W.; Petrone, A.; Li, X.; Cossairt, B. M. *Journal of the American Chemical Society* **2016**, *138*, 1510–1513.

(23) Ge, J.; Liang, J.; Chen, X.; Deng, Y.; Xiao, P.; Zhu, J.-J.; Wang, Y. *Chemical Science* **2022**, *13*, 11755–11763.

(24) Goodman, J.; Weare, J. *Communications in Applied Mathematics and Computational Science* **2010**, *5*, 65–80.

(25) Green, M. L. H. *Journal of Organometallic Chemistry* **1995**, *500*, 127–148.

(26) Guyot-Sionnest, P.; Shim, M.; Matranga, C.; Hines, M. *Phys. Rev. B Condens. Matter* **1999**, *60*, R2181–R2184.

(27) Hartley, C. L.; Kessler, M. L.; Dempsey, J. L. *Journal of the American Chemical Society* **2021**, *143*, 1251–1266.

(28) Hemmert, C.; Verelst, M.; Tuchagues, J.-P. *Chemical Communications* **1996**, *0*, 617–618.

(29) Houtepen, A. J.; Hens, Z.; Owen, J. S.; Infante, I. *Chemistry of Materials* **2017**, *29*, 752–761.

(30) Hughes, K. E.; Stein, J. L.; Friedfeld, M. R.; Cossairt, B. M.; Gamelin, D. R. *ACS Nano* **2019**, *13*, 14198–14207.

(31) Jang, E.; Kim, Y.; Won, Y.-H.; Jang, H.; Choi, S.-M. *ACS Energy Letters* **2020**, *5*, 1316–1327.

(32) Kambhampati, P. *Journal of Physical Chemistry C* **2011**, *115*, 22089–22109.

(33) Kessler, M. L.; Starr, H. E.; Knauf, R. R.; Rountree, K. J.; Dempsey, J. L. *Physical Chemistry Chemical Physics* **2018**, *20*, 23649–23655.

(34) Kilina, S.; Ivanov, S.; Tretiak, S. *J. Am. Chem. Soc.* **2009**, *131*, 7717–7726.

(35) Kirkwood, N.; Monchen, J. O. V.; Crisp, R. W.; Grimaldi, G.; Bergstein, H. A. C.; du Fossé, I.; van der Stam, W.; Infante, I.; Houtepen, A. J. *Journal of the American Chemical Society* **2018**, *140*, 15712–15723.

(36) Klein, M. D.; Bisted, C. H.; Dou, F. Y.; Sandwisch, J. W.; Cossairt, B. M.; Khalil, M. *J. Phys. Chem. C* **2023**, *127*, 16970–16978.

(37) Knauf, R. R.; Lennox, J. C.; Dempsey, J. L. *Chem. Mater.* **2016**, *28*, 4762–4770.

(38) Kwon, H.; Kim, S.; Kang, S. B.; Bang, J. *CrystEngComm* **2022**, *24*, 3724–3730.

(39) Kwon, Y.; Kim, S. *NPG Asia Materials* **2021**, *13*, 1–16.

(40) Kwon, Y.; Oh, J.; Lee, E.; Lee, S. H.; Agnes, A.; Bang, G.; Kim, J.; Kim, D.; Kim, S. *Nat. Commun.* **2020**, *11*, 3127.

(41) Le Feber, B.; Prins, F.; De Leo, E.; Rabouw, F. T.; Norris, D. J. *Nano Letters* **2018**, *18*, 1028–1034.

(42) Leger, J. D.; Friedfeld, M. R.; Beck, R. A.; Gaynor, J. D.; Petrone, A.; Li, X.; Cossairt, B. M.; Khalil, M. *Journal of Physical Chemistry Letters* **2019**, *10*, 1833–1839.

(43) Matsakis, D.; Coster, A.; Laster, B.; Sime, R. *Physics Today* **2019**, *72*, 10–11.

(44) Meitner, L. *Zeitschrift für Physik* **1922**, *9*, 145–152.

(45) Mule, A. S.; Mazzotti, S.; Rossinelli, A. A.; Aellen, M.; Prins, P. T.; van der Bok, J. C.; Solari, S. F.; Glauser, Y. M.; Kumar, P. V.; Riedinger, A.; et al. *J. Am. Chem. Soc.* **2021**, *143*, 2037–2048.

(46) Nguyen, K. A.; Pachter, R.; Day, P. N. *Journal of Physical Chemistry A* **2020**, *124*, 10472–10481.

(47) Owen, J. *Science* **2015**, *347*, 615–616.

(48) Owen, J. S.; Park, J.; Trudeau, P. E.; Alivisatos, A. P. *Journal of the American Chemical Society* **2008**, *130*, 12279–12281.

(49) Park, N.; Monahan, M.; Ritchhart, A.; Friedfeld, M. R.; Cossairt, B. M. *J. Vis. Exp.* **2019**, *2019*, 1–8.

(50) Pearce, O. M.; Duncan, J. S.; Damrauer, N. H.; Dukovic, G. *Journal of Physical Chemistry C* **2018**, *122*, 17559–17565.

(51) Puzder, A.; Williamson, A. J.; Zaitseva, N.; Galli, G.; Manna, L.; Alivisatos, A. P. *Nano Letters* **2004**, *4*, 2361–2365.

(52) Rempel, J. Y.; Trout, B. L.; Bawendi, M. G.; Jensen, K. F. *Journal of Physical Chemistry B* **2006**, *110*, 18007–18016.

(53) Ricci, F.; Marougail, V.; Varnavski, O.; Wu, Y.; Padgaonkar, S.; Irgen-Gioro, S.; Weiss, E. A.; Goodson T., 3. *ACS Nano* **2021**, *15*, 12955–12965.

(54) Ritchhart, A.; Cossairt, B. M. *Inorganic Chemistry* **2019**, *58*, 2840–2847.

(55) Ritchhart, A.; Cossairt, B. M. *Angew. Chem.* **2018**, *130*, 1926–1930.

(56) Roh, J.; Park, Y.-S.; Lim, J.; Klimov, V. I. *Nature Communications* **2020**, *11*, 271.

(57) Rotem, M.; Goldberg, I.; Shmueli, U.; Shvo, Y. *Journal of Organometallic Chemistry* **1986**, *314*, 185–212.

(58) Schnitzenbaumer, K. J.; Labrador, T.; Dukovic, G. *Journal of Physical Chemistry C* **2015**, *119*, 13314–13324.

(59) Socrates, G., *Infrared and Raman Characteristic Group Frequencies: Tables and Charts*; John Wiley & Sons: 2004.

(60) Soloviev, V. N.; Eichhöfer, A.; Fenske, D.; Banin, U. *J. Am. Chem. Soc.* **2001**, *123*, 2354–2364.

(61) Stelmakh, A.; Aebli, M.; Baumketner, A.; Kovalenko, M. V. *Chemistry of Materials* **2021**, *33*, 5962–5973.

(62) Su, E.; Guven, A.; Kani, I. *Applied Organometallic Chemistry* **2018**, *32*, e4105.

(63) Swenson, N. K.; Ratner, M. A.; Weiss, E. A. *J. Phys. Chem. C* **2016**, *120*, 6859–6868.

(64) Taheri, P.; Wielant, J.; Hauffman, T.; Flores, J. R.; Hannour, F.; De Wit, J. H. W.; Mol, J. M. C.; Terryn, H. *Electrochimica Acta* **2011**, *56*, 1904–1911.

(65) Utzat, H.; Sun, W.; Kaplan, A. E. K.; Krieg, F.; Ginterseder, M.; Spokoyny, B.; Klein, N. D.; Shulenberger, K. E.; Perkinson, C. F.; Kovalenko, M. V., et al. *Science* **2019**, *363*, 1068–1072.

(66) Van Der Stam, W.; Du Fossé, I.; Grimaldi, G.; Monchen, J. O. V.; Kirkwood, N.; Houtepen, A. J. *Chemistry of Materials* **2018**, *30*, 8052–8061.

(67) Weiss, E. A. *ACS Energy Letters* **2017**, *2*, 1005–1013.

(68) Weiss, E. A. *Accounts of Chemical Research* **2013**, *46*, 2607–2615.

(69) Won, Y.-H.; Cho, O.; Kim, T.; Chung, D.-Y.; Kim, T.; Chung, H.; Jang, H.; Lee, J.; Kim, D.; Jang, E. *Nature* **2019**, *575*, 634–638.

(70) Wu, K.; Li, H.; Klimov, V. I. *Nature Photonics* **2018**, *12*, 105–110.

(71) Xiao, P.; Zhang, Z.; Ge, J.; Deng, Y.; Chen, X.; Zhang, J.-R.; Deng, Z.; Kambe, Y.; Talapin, D. V.; Wang, Y. *Nature Communications* **2023**, *14*, 49.

(72) Xie, L.; Zhao, Q.; Jensen, K. F.; Kulik, H. J. *Journal of Physical Chemistry C* **2016**, *120*, 2472–2483.

(73) You, Y.; Tong, X.; Wang, W.; Sun, J.; Yu, P.; Ji, H.; Niu, X.; Wang, Z. M. *Advanced Science* **2019**, *6*, 1801967.

(74) Zeleňák, V.; Vargová, Z.; Györyová, K. *Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy* **2007**, *66*, 262–272.

(75) Zhang, J.; Zhang, H.; Cao, W.; Pang, Z.; Li, J.; Shu, Y.; Zhu, C.; Kong, X.; Wang, L.; Peng, X. *Journal of the American Chemical Society* **2019**, *141*, 15675–15683.

(76) Zhao, Q.; Kulik, H. J. *Chemistry of Materials* **2018**, *30*, 7154–7165.

Chapter 3

MECHANISTIC INSIGHTS OF LIGAND POPULATION CHANGE IN CADMIUM SULFIDE QUANTUM DOT GELATION VIA TEMPERATURE-DEPENDENT FTIR SPECTROSCOPY

Quantum dot (QD)-based gels offer a promising platform for a variety of applications, ranging from gas adsorption to nanowire synthesis. In this study, we employed temperature-dependent Fourier Transform Infra Red Spectroscopy(TD-FTIR) and proton nuclear magnetic resonance ^{1}H NMR to investigate and compare the surface ligand populations of CdS quantum dot (QD) solutions washed with MeCN and EtOH during gelation. Our findings reveal that the MeCN-washed QDs contained a significant amount of $Cd(Ol)_2$ impurities, which play a critical role in promoting gelation. In contrast, the EtOH-washed CdS QDs primarily exhibited X-type ligands in both chelating and bridging binding modes, as confirmed through ICP and FTIR. Further heating studies of $Cd(Ol)_2$ were performed to explore gelation dynamics, showing that short heating times result in the formation of a physical gel, while prolonged heating leads to irreversible decarboxylation into ketonic products that scaffold a chemical gel.

3.1 Introduction

Colloidal quantum dots (QDs) have garnered significant research interest due to their unique size and morphology dependent optical and electronic properties[55,57,60] . Offering versatility in a range of optical applications such as lasing[37,58] , light emitting diodes[25,38] , and imaging[11] , QDs are enticing to use in a variety of devices due to their narrow emission line widths, strong absorption, and solution based synthetic

size tunability[10,30,67] . In this regard, macrostructures of QDs, such as quantum dot superlattices, films, and gels, are gaining attention for their ability to integrate the unique properties of individual QDs into collective, enhanced functionalities.[3,35,52,64] QD macrostructures overcome issues associated with a single QD such as inconsistent performance[43] and stability,[2,24,56] and the coupling of optoelectronic properties between adjacent QDs allows for emergent behaviors in charge mobility and enhanced photoluminescence not previously seen in isolated dots[15,31] .

The identity of surface ligands plays a crucial role in defining the chemical properties of quantum dots, as the ligand tail and surface anchor determines solvent compatibility and imparts specific chemical functionalities.[5,9,13,27,65] Ligands also provide charge stabilization to the non stoichiometric surface of the quantum dot core,[49,50] and can influence overall electronic and optical behavior[34] . In oleate-capped QDs, the ligands attach to the surface through carboxylate head groups, which exhibit distinct infrared spectral signatures detectable by FTIR spectroscopy[62] . Previously, ligand populations have been quantified by fitting these FTIR signatures to Gaussian functions, offering insights into surface chemistry and ligand distribution.[27,50,62] Similarly, these ligands can occur in various bonding environments based on their bond type including Z-type , L-type, and X-type.[20]

QD gels have emerged as a promising new class of quantum dot (QD) macrostructures, offering unique advantages for various applications. Li et al.[33] describe that the self healing nature and adhesive properties of QD embedded hydro-gels is beneficial in DNA labeling, and Korala[29] et al. explain that QD sol-gel based thin films offer increased charge transfer due to increased interfacial contact from the 3 dimensional gel network in comparison to non-gel based application methods. While QD-embedded gels hold promise for a wide range of applications, their synthesis traditionally relies on either ligand exchange to achieve compatibility with hydrogels[8,16,35] or oxidation of surface anions to promote gelation[18,24,52] both of which intrinsically alter the surface chemistry of the QD. In QD embedded hydrogels, QDs often require rigorous

ligand exchange procedures for water based gel matrix compatibility.[16,35,59] Furthermore, surface oxidation of anions on the QD surface intrinsically alters the surface chemistry of the QD.

In-situ gel formation in non-polar solvents is a burgeoning problem in semiconductor QD synthesis.[15,54,61] Both Enomoto[15] and Welsch[61] note that Cd oleate , which is a common cadmium precursor and QD ligand , polymerizes into a gel. Furthermore, there are many examples of QD solution viscosity change and 3D structure formation based on oleate ligand and metal oleate precursor concentration[17,36,41] in QD solutions. However, little is known about the surface chemistry of QDs in these non-polar solvent based gels, especially regarding how surface ligand binding motif populations influence the gelation process. Additionally, the interactions during gel formation between surface ligands and the surrounding solvent within the gel matrix remain poorly understood.

Given the wide range of applications for quantum dots, gaining insights into optimal synthetic procedures tailored to specific functions is essential for maximizing their performance. We seek to understand ligand dependent gel formation of CdS oleate QDs in non-polar solvents through investigation of their infrared spectra over the course of gelation. Using TD-FTIR and ^{1}H NMR, we confirm that excess Cd oleate precursor forms a gel in oleate capped CdS QD solutions. In addition, we show that washing CdS with ethanol prevents gelation due to removal of weakly bound Z-type $Cd(Ol)_2$ and unbound Cd oleate. We also show that increased temperature promotes the shift of covalently bound oleate from the chelating moiety to a bridging moiety. Finally, we show that this shift in ligand populations prompts the formation of both a physical and a chemical gel.

3.2 Methods

Synthesis of CdS-400nm Quantum Dots Cadmium oleate was synthesized according to a previously reported procedure.[22]

Synthesis of CdS-400nm Quantum Dots CdS-400 quantum dots were synthesized according to a previously reported procedure.[21] In short, cadmium oleate (3.6 mmol, 2.43 g) and hexadecane (120 mL) are combined under nitrogen in a 250 mL, 3-neck round bottom flask. The flask is then heated to 80°C under vacuum for 1 hour after which nitrogen is backfilled and the temperature is raised to 190°C. Immediately after complete cadmium oleate dissolution, tetramethyl thiourea (3 mmol, 0.396 g) in 74% diphenyl ether and 26% biphenyl eutectic mixture (3 mL) is promptly injected. After injection, the temperature is quickly ramped to 240°C. While the flask is heating, aliquots are taken to monitor the quantum dot growth. Once the desired wavelength is reached (λ_{max} = 400nm), the reaction is immediately removed from heat and cooled to 90°C at which point the hexadecane is removed by vacuum distillation. Inside a nitrogen-filled glovebox, the final yellow pellet is dissolved in minimal toluene. A byproduct of this reaction is a CdS nanocluster species (λ_{max} = 323nm) which is removed by size-selective precipitation with ethyl acetate. With the quantum dots isolated, they are purified by precipitation and redissolution three times using acetonitrile and toluene respectively. The final product is stored as a white solid. For the ethanol precipitation, the final product was resuspended in minimal toluene and precipitated once with ethanol.

Temperature Dependent FTIR FTIR measurements were conducted in a Jasco 4700 FTIR spectrophotometer under a positive pressure of N_2. Spectra were averaged 32 times with data taken every 1 cm^{-1}. Sample was flowed through a liquid cell with a 620 μm Teflon spacer. Heated measurements were conducted with a temperature- controlled demountable liquid cell (Harrick) paired with a Peltier heater (Harrick). In flowed experiments, the sample was heated 1°C/min while flowing, and left to equilibrate for 5 minutes at each temperature point. Flowing was stopped when recording spectra and then resumed to ensure thermal equilibrium. For stationary experiments, the sample was heated at 0.5°C /min. In each experiment type, measurements were taken every 5°C.

NMR Spectroscopy: ^{1}H-NMR spectra were recorded on a Bruker AVANCE-III spectrometer (500 MHz). All spectra were recorded employing standard pulse techniques. Samples were dissolved in toluene-d8 and chemical shifts were measured relative to residual solvent (toluene-d8 at 2.08 ppm).

Inductively Coupled Plasma Optical Emission Spectroscopy: ICP-OES was conducted on a Perkin Elmer Optima 8300 inductively couple plasma – optical emission spectrophotometer. For sample preparations, a small amount of each sample (less than 10 mg) was digested overnight in 1:1 70 % nitric acid/hydrogen peroxide, diluted until the concentration of nitric acid was 2 %, and filtered through a Nylon microsyringe filter. Calibration curves were prepared using the appropriate standard and from 2 % nitric acid stock solution in 18 MΩ deionized water.

3.3 Results and Discussion

II-IV quantum dots (QDs), such as CdS, are typically stabilized with long-chain fatty acid ligands, where carboxylate groups serve as the primary binding species.[6,27,50] In the case of CdS-oleate, oleate ligands often form Z-type complexes, where two oleate molecules coordinate with a cadmium ion as $Cd(Ol)_2$.[7,13] These Z-type complexes can readily accept electron pairs from surface anions, thereby contributing to the stabilization of the QD surface.[7,20] In addition, L-type oleic acid may also interact with the surface, acting as an electron pair donor to displace Z-type oleates[20] . Finally, X-type oleate ligands can bind covalently to the quantum dot surface, further contributing to its overall stabilization by balancing charge and enhancing surface passivation[53] .

Instances of gelation have been observed following hot-injection synthesis of PbS and CdS quantum dots at temperatures above 100°C. Similarly, Welsch and Shi successfully synthesized gels by preparing Cd oleate from CdO and heating the resulting Cd oleate above 100°C for at least 15 minutes, then allowing it to cool to room temperature. To investigate surface ligand identity and relative binding populations, temperature-dependent FTIR spectroscopy was performed, with measurements taken

every 5°C from 25°C to 110°C. A 4 mg/mL solution of CdS -oleate in tetrachloroethylene was introduced into the sample chamber, where it was heated at a controlled rate of 1°C every 2 minutes. This approach enables monitoring of changes in ligand binding and surface chemistry as a function of temperature and by extension gel formation. Combining TD-FTIR with TD-^{1}H NMR allows us to monitor ligand population distributions and dynamic changes of X and Z-type populations that dissociate and drive gelation.

Z-type ligands, with their ability to be displaced and diffuse throughout the system, likely facilitate physical gelation.[28,61] Their interaction with metal ions provides a dynamic scaffolding that enables physical crosslinking and promotes reversible interactions among ligands.[16] The inclusion of CdS quantum dots (QDs) in the solution can further enhance this scaffolding effect, as the QDs introduce additional coordination sites and increase the overall complexity of the network.[57]

In contrast, X-type ligands of oleate, which form covalent bonds often exhibit binding motifs that bridge multiple metal centers. These bridging interactions are particularly suited to driving the formation of chemical gels, where permanent or semi-permanent crosslinks result in a more robust and chemically stable gel network.[24]

To determine whether excess cadmium oleate influences physical gel formation, two different washing procedures were performed and analyzed using FTIR. In the first procedure, CdS-Oleate quantum dots (QDs) were precipitated three times with acetonitrile (MeCN). In the second procedure, the CdS-Oleate QDs were precipitated three times with MeCN, followed by a final wash with ethanol to remove any remaining excess Cd oleate. The resulting TD-FTIR spectra of both washing procedures is shown in Figure 3.1.

After heating to 110°C, the MeCN washed CdS-oleate QDs passes the positive inversion test indicating formation of a 3 dimensional macro-structures and gelation. It should be noted that simply leaving MeCN washed CdS-oleate QDs at room temperature for seven days resulted in failure of the inversion test and no gel formation

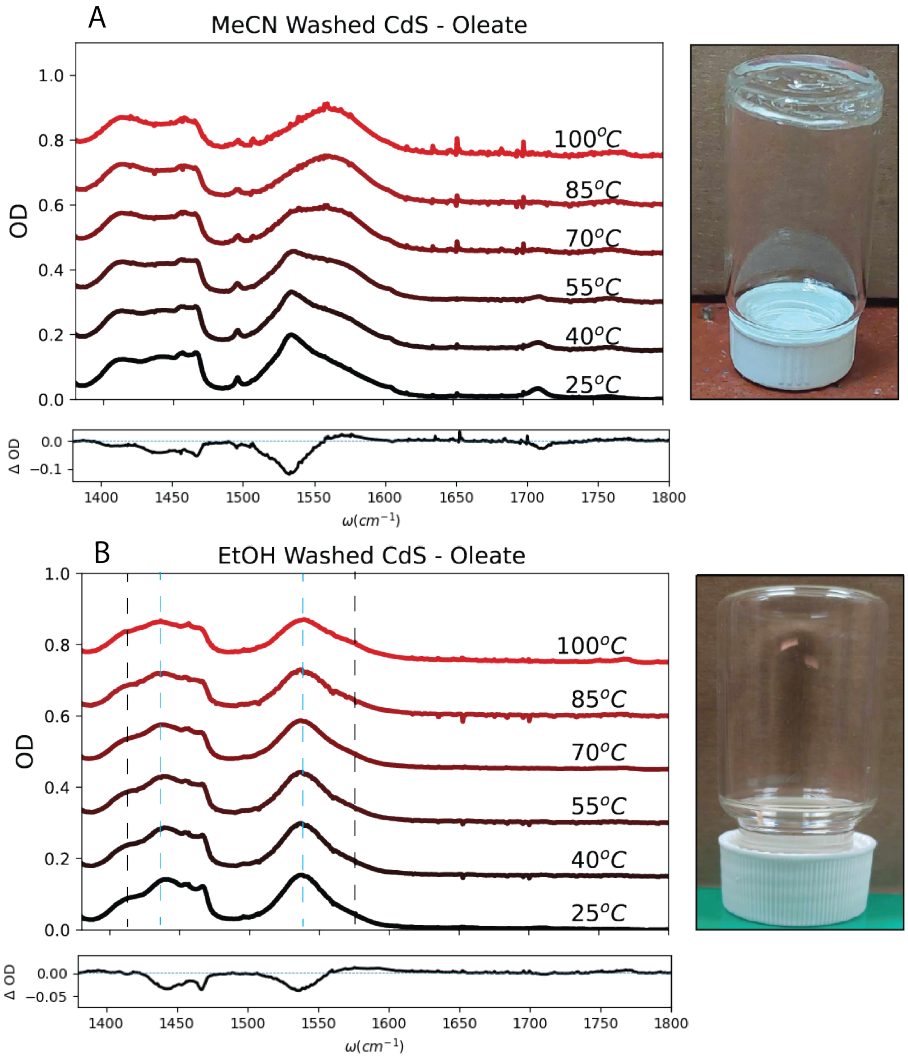

Figure 3.1: TD-FTIR spectra of CdS-Oleate QDs washed with MeCN (a) and washed with EtOH (b), with intervals of every 15°C from 25°C to 100°C shown, with difference plots of 100°C and 25°C . Adjacent to each spectrum is the corresponding inversion test of 4 mg/mL QD solutions after 7 days of resting at room temperature.

(Figure B.1). Likewise, EtOH washed CdS-oleate QDs fail the inversion test (Figure 3.1) indicating no gel formation.

Figure 3.1 also shows the TD-FTIR spectra of carboxylate vibrations of bound oleate (1400 -1620 cm $^{-1}$) and free oleic acid ligand (1700 -1780 cm $^{-1}$). Since the carboxylate head group of oleate is bidentate, there are a variety of possible binding modes, such as chelating, syn-anti bridging, syn-syn bridging, monodentate, and dative coordination.[27,32,68] Each of these binding motifs exhibits a symmetric stretch between $1400\,\mathrm{cm}^{-1}$ and $1500\,\mathrm{cm}^{-1}$, and an asymmetric stretch between $1500\,\mathrm{cm}^{-1}$ and $1620\,\mathrm{cm}^{-1}$. While the CH_2 bending modes from the oleate tail overshadow the symmetric stretching region (Figure B.2), we can still observe general shape changes in this region as the temperature increases. To spectrally distinguish between different binding motifs, it is common to use the method of peak separation established by Zelenák et al.,[68] in combination with X-ray crystallography. In the method of peak separation, $\Delta\omega = \omega_{\mathrm{asym}} - \omega_{\mathrm{sym}}$ is used to characterize binding motifs spectrally. The typical order is as follows:

$$\Delta\omega_{\text{chelating}} < \Delta\omega_{\text{syn-anti bridging}} < \Delta\omega_{\text{syn-syn bridging}} < \Delta\omega_{\text{monodentate}},$$

reflecting variations in electron density over the O-C-O bond.

For the ethanol-washed CdS-Oleate particles, a peak is observed in the symmetric stretching region at $1435\,\mathrm{cm}^{-1}$ and another at $1405\,\mathrm{cm}^{-1}$. There is no absorbance peak associated with oleic acid (Figure B.2) at $1435\,\mathrm{cm}^{-1}$, so likely the $1435\,\mathrm{cm}^{-1}$ is associated with a symmetric C-O binding moiety vibration. Compared to the CH_2 bend at $1468\,\mathrm{cm}^{-1}$ the $1405\,\mathrm{cm}^{-1}$ peak increases over temperature while the vibration at $1412\,\mathrm{cm}^{-1}$ decreases and broadens. Thus, $1405\,\mathrm{cm}^{-1}$ peak is likely associated with an additional symmetric C-O binding moiety. Similarly, in the asymmetric stretching region, peaks were found at $1536\,\mathrm{cm}^{-1}$ and $1570\,\mathrm{cm}^{-1}$. These peaks appear to broaden as the temperature increases. The observed peaks correspond to $\Delta\omega$ values of $97\,\mathrm{cm}^{-1}$ and $165\,\mathrm{cm}^{-1}$, respectively. With these fit parameters , we find that the chelating moiety is 78% ± 6% of the surface ligand binding population with the other 78% ± 6% of ligand head groups in the bridging moiety.

$\Delta\omega$ values between $40\,cm^{-1}$ and $110\,cm^{-1}$ are characteristic of chelating moieties, while $\Delta\omega$ values between $120\,cm^{-1}$ and $180\,cm^{-1}$ correspond to bridging motifs.[32,68] While others have successfully spectrally distinguished syn-syn and syn-anti bridging modes, they focused on either pure metal carboxylates[68] or atomically precise, magic-sized clusters,[32,49] which offer sharper spectral signatures than our more complex QD system. The facile surface of CdS - QDs offer many environments for carboxylate to bond likely broadening the spectral line width and promoting multiple vibrational frequencies. Cass et al. reported comparable FTIR spectra when washing their PbS QDs with methanol, attributing a symmetric stretch at $1404\,cm^{-1}$ and an asymmetric stretch at $1525\,cm^{-1}$ to a chelating moiety ($\Delta\omega = 120\,cm^{-1}$), and a symmetric stretch at $1401\,cm^{-1}$ with an asymmetric stretch at $1549\,cm^{-1}$ to bridging moiety. As such, we ascribe 2 binding moieties to the EtOH washed CdS-oleate QDs - chelating and bridging, and the spectra can successfully be fit to 2 gaussians (Figure 3.2), with fit parameters in Table 3.1.

Identity	**Center (cm^{-1})**	**Width**	**Area %**
Chelating	1536 ± 1	16 ± 1	$78\% \pm 6\%$
Bridging	1573 ± 6	20 ± 5	$22\% \pm 6\%$

Table 3.1: Widths, centers, and area populations of the surface oleates on EtOH washed CdS QDs

The MeCN-washed QDs exhibit similar peak trends in the symmetric stretching region. Specifically, the $1438\,cm^{-1}$ peak decreases in amplitude with increasing temperature, relative to the aliphatic stretch at $1468\,cm^{-1}$. Additionally, the $1405\,cm^{-1}$ peak shows an increase in intensity as the temperature rises. In the asymmetric stretching region, a sharp peak at $1530\,cm^{-1}$ diminishes and eventually disappears with increasing temperature, while a new vibrational peak at $1560\,cm^{-1}$ emerges and

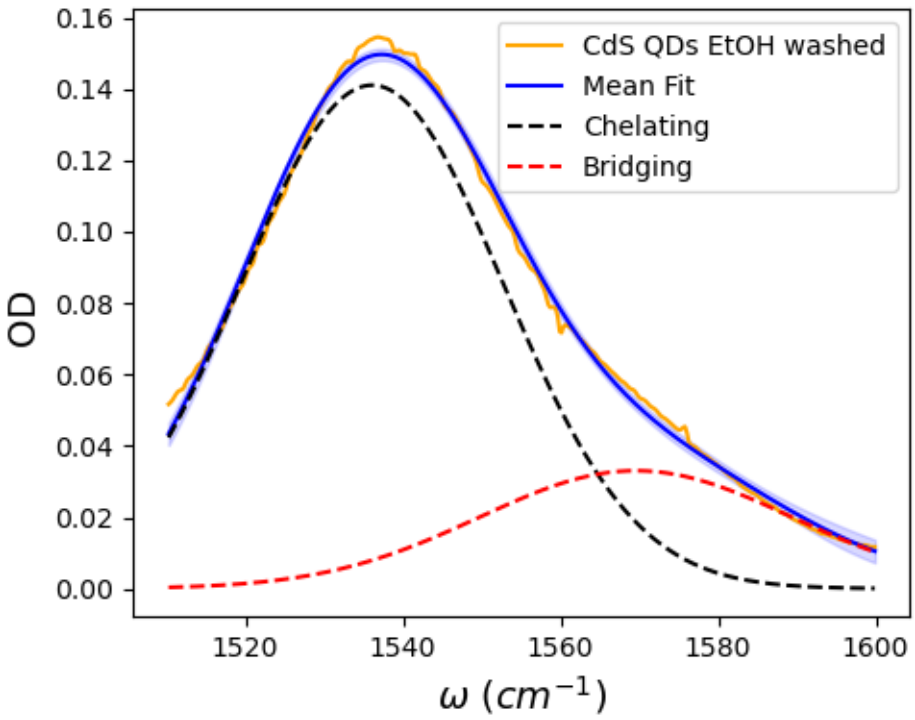

Figure 3.2: FTIR of EtOH washed CdS QDs at 25°C fit to 2 gaussians associated with a chelating and bridging binding motif respectively.

grows in intensity. Unlike the EtOH-washed QDs, the MeCN-washed QDs show the presence of free oleic acid in the room temperature solution, as indicated by a peak at 1710 cm^{-1}. This peak gradually disappears upon heating and is no longer observed by 85 °C.

The similar temperature-dependent trends observed for the peaks at 1438 cm^{-1} and 1530 cm^{-1} and 1405 cm^{-1} and 1560 cm^{-1} respectively suggest that each peak pair originate from the same binding moiety. It is easy to simply assume that these peak pairs are associated with the chelating and bridging binding moiety respectively as in the EtOH washed example. However, the 1530 cm^{-1} peak has been associated with $Cd(Ol)_2$ on numerous occasions.[15,54,61] FTIR does not whether the oleate ligand is

covalently bound to the QD or to a Cd ion in Cd Oleate. Put simply FTIR cannot distinguish between Z-type and X-type carboxylate with the same covalently bound moiety. For this , ^{1}H NMR is required.

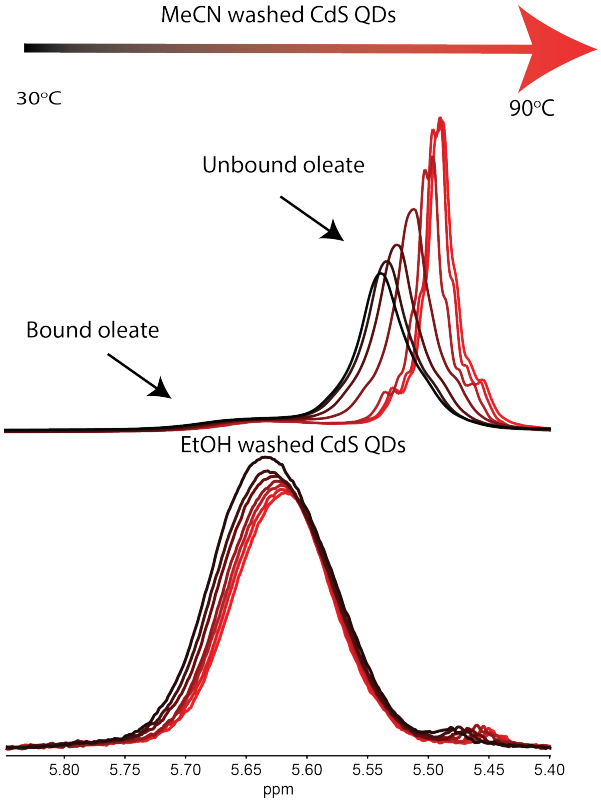

Figure 3.3: TD ^{1}H NMR of the vinyl oleate hydrogens in CdS-Oleate QDs washed with MeCN (a) and washed with EtOH (b) at and every 10°C from 30°C to 90°C.

Figure 3.3 shows the ^{1}H NMR spectra centered on the vinyl proton region from temperature-dependent measurements of CdS-oleate QDs washed with MeCN and EtOH. The vinyl protons are spectrally isolated from other proton peaks and allow for differentiation between bound and unbound oleate species. Bound carboxylates are characterized by broad vinyl peaks between 5.60 and 5.70 ppm,[7,40] whereas unbound carboxylates appear further upfield, ranging from 5.51 to 5.40 ppm.[28] Both $Cd(Ol)_2$

(the desorbed Z-type ligand) and L-type oleic acid are classified as unbound and remain spectrally distinct across all temperatures studied, as shown in Figure B.3. At low temperatures, $Cd(Ol)_2$ exhibits a broad peak centered around 5.51 ppm, while oleic acid shows a peak centered at 5.41 ppm. The $Cd(Ol)_2$ peak sharpens into an octuplet and shifts upfield with increasing temperature. This peak sharpening is likely due to increased molecular motion that averages out anisotropy induced by cd-o bonds that contributes to peak broadening.[4] Similarly, the oleic acid octuplet is evident at all temperatures and also shifts upfield as the temperature rises.

Both ^{1}H NMR spectra for the MeCN-washed and EtOH-washed CdS QDs show evidence of bound and unbound carboxylates, but in significantly different ratios. In both samples, the unbound species appear as broad peaks at low temperatures, transitioning into sharp octuplets at higher temperatures, following a similar temperature-dependent behavior as observed for $Cd(Ol)_2$ in Figure B.3. This suggests that both QD solutions contain $Cd(Ol)_2$, albeit at varying concentrations in Table 3.2.

Table 3.2: Comparison of Cd:S ratios from ICP and Bound:Unbound 90°C for MeCN and EtOH Washed CdS-Oleate QDs

	Cd:S Ratio	Bound:Unbound Carboxylate (90°C)
MeCN Washed CdS-Oleate QDs	3.81:1	1:9
EtOH Washed CdS-Oleate QDs	4.91:1	42:1

Understanding core-surface chemistry is essential for elucidating ligand dynamics in nanomaterials. Table 3.2 presents the cadmium-to-sulfur (Cd:S) ratio of CdS QDs based on the washing solvent, as well as the ratio of bound to unbound ligands at 90 °C, determined via ^{1}H NMR peak integration. The bound-to-unbound ratio was specifically analyzed at 90 °C due to largened peak separation between the bound and unbound regions in the NMR spectra.

The EtOH-washed CdS QDs exhibit a significantly higher ratio of bound ligand to unbound ligand, with a ratio of 42:1, compared to their MeCN-washed counterparts at 1:9. Additionally, the EtOH-washed QDs demonstrate a more Cd-rich core, which aligns with the expected behavior of alcohols acting as L-type ligands to readily displace Z-type cadmium oleate, leading to a Cd-rich surface.[63] According to Hartley and colleagues,[23] metal-rich surfaces are often passivated by X-type ligands for charge stabilization.

Consequently, the bound oleate in the EtOH-washed CdS QDs is likely X-type, with FTIR spectra in Figure 3.2 indicating that oleate vibrations from $Cd(Ol)_2$ are overshadowed by those of QD-bound X-type oleate. Thus, we propose that the surface of EtOH-washed CdS QDs is predominantly ligated by X-type oleate, exhibiting chelating and bridging binding modes.

The MeCN-washed CdS QDs exhibit a significantly higher proportion of unbound ligands relative to bound ligands, with an approximate ratio of 9:1. Notably, the unbound region in the TD-^{1}H NMR spectra of these QDs closely resembles that of $Cd(Ol)_2$ suggesting that the unbound oleate species in this sample predominantly exists in the form of $Cd(Ol)_2$. This high concentration of $Cd(Ol)_2$ impurity agrees Welsch's assertion that heating of $Cd(Ol)_2$ causes gelation.[61] Furthermore, this high unbound ligand to bound ligand ratio likely skews the atomic ratio of Cd:S found through ICP, such that the ratio supplies little information about core surface chemistry. Consequently, discerning the specific temperature-dependent behavior of the bound ligand versus unbound $Cd(Ol)_2$ becomes challenging, as the oleate associated with the $Cd(Ol)_2$ impurity produces overlapping peaks within the same infrared spectral region. Therefore, a more detailed investigation into the temperature dynamics of $Cd(Ol)_2$ is warranted to fully elucidate the binding environment.

It is worth noting that the Cd ratio determined via ICP for both MeCN-washed and EtOH-washed QDs is unusually high for CdS QDs of this size.[48] Typically, this ratio hovers around 2:1; however, in this case, the observed ratios are approximately dou-

ble that value. For the MeCN-washed particles, this can be reasonably attributed to the presence of excess $Cd(Ol)_2$ in solution. However, the elevated ratio for the EtOH-washed particles is more puzzling, given the lack of evidence for diffused $Cd(Ol)_2$ in this system. A likely explanation is incomplete sulfur digestion during sample preparation, which could artificially inflate the Cd ratios.

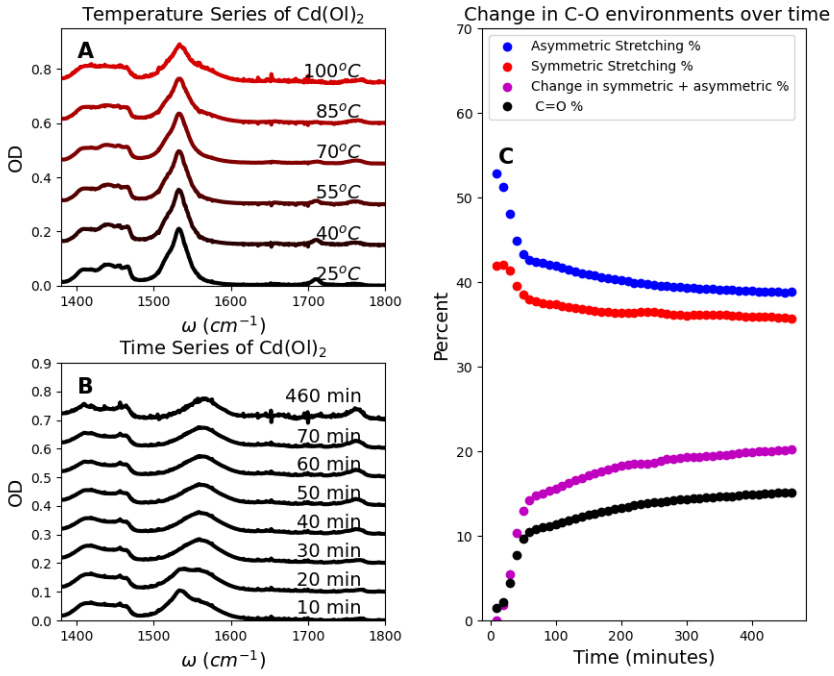

Figure 3.4: A) TD-FTIR of $Cd(Ol)_2$ in tetrachloroethylene with temperatures from 25°C to 100°C in increments of 15°C B) a time series of $Cd(Ol)_2$ at 110°C with spectra taken every 10 minutes and a final spectra at 460 minutes C) percent area of the symmetric C-O stretching peaks, asymmetric C-O stretching peaks, and C=O stretching peaks in comparison to the total carboxylate region.

Figure 3.4A shows the TD-FTIR spectra of $Cd(Ol)_2$ from 25°C to 100°C in increments of 15°C. Given that the TD-FTIR spectra of MeCN-washed CdS in Figure

3.1 predominantly consists of $Cd(Ol)_2$, both samples exhibit similar temperature-dependent behavior. Specifically, there is a reduction in the peak at 1530 cm^{-1} and the emergence of a new peak at 1560 cm^{-1}. We have previously attributed these peaks to the chelating and bridging binding moieties, respectively. These metal-organogels are formed due to solvent molecules being physically trapped between long-chain ligands[15,54] as a result of london dispersion forces between the ligand tales in the form of a physical organogel.[61] Definitionally the bridging binding moiety connects two Cd atoms, which allows for an increase in intermolecular interaction of the alkyl tails with each other as well as the solvent.[15] This is further supported by the high reversibility observed in the FTIR spectra during heating and cooling cycles (Figure B.5). At higher temperatures, increased molecular motion enhances the likelihood of ligand tail interactions, but these interactions are non covalent due to lack of FTIR permanence. Physical gels are more sensitive to thermal motion than chemical gels, so the change of vibrational populations based on temperature is to be expected.[1,26]

We observe the complete disappearance of the chelating peak in the MeCN-washed QD sample upon heating to 110°C where it is still present for $Cd(Ol)_2$. This is likely due to a combination of two factors. First, the oleate ligand can bond to cadmium X-type binding sites on the nanocrystal surfaces as well as to solvated Cd ions. Given that cadmium originates from two different sources, the oleate head group is inherently required to adopt a bridging configuration. Second, the lower concentration of $Cd(Ol)_2$ in solution likely influences the dynamics of the chelating-to-bridging moiety transition.

We also observe the formation of a chemical gel when the $Cd(Ol)_2$ is heated longer than an hour. Welsch[61] and Enomoto[15] describe synthetic procedures that require heating above 100°C for anywhere between 15 minutes and multiple hours, but there is contestation over whether these gels have the same macrostructure. Therefore, we measure time-dependent $Cd(Ol)_2$ binding motif changes by holding a 2 mg/mL solution of $Cd(Ol)_2$ at 110°C for 460 minutes (Figure 3.4 B). By 70 minutes into

this heating, the entire surface ligand head group population shifts from chelating to bridging. Additionally, the vibrational peak at 1760 cm^{-1} increases. Shi et al.[54] demonstrate that $Cd(Ol)_2$ gels adopt a bamboo-like structure, with layers of $Cd_2(Ol)_4$ stacked on top of one another. As such, the increase in bridging moiety population promoting the formation of $Cd_2(Ol)_4$ layers.

The transition of the chelating binding moiety into other species is of particular interest. Figure 3.4C illustrates how the percent areas of the asymmetric C-O , symmetric C–O stretching regions, and C=O stretching change with temperature relative to the total area of all C–O and C=O stretching regions.

For this analysis, we assume that an oscillator cannot simultaneously act as a free C=O ligand and a bound C–O carboxylate. This assumption allows us to track how ligands transition between being bonded to cadmium in $Cd(Ol)_2$ and existing in a free state over time. Similarly, when finding the percent area difference of the symmetric region at 110°C over time, we assume that the area associated with the CH bending modes of oleic acid does not change over time. In both the symmetric and asymmetric stretching regions for bound carboxylates, we observe a sharp decrease in relative population at initial temperatures, likely due to the reduction of the chelating binding moiety. Interestingly, the percent area of the C=O stretching region increases sharply. Eventually, the percent areas of both free and bound ligands plateau, suggesting that continued heating leads to the conversion of a portion of $Cd(Ol)_2$.

To determine whether the lost population of bonded carboxylates fully converts into free ligand or some other reaction product, we calculate the difference between the percent population of bonded carboxylates at the initial time point and at each subsequent time point. If this difference at any given time matches the relative percentage of free ligand in the C=O region, we can infer that the bonded carboxylates are entirely converted into free ligand. While the area of symmetric C-O stretching region encompasses CH bending from the alkyl tail of oleate , the assumption that the CH area does not change over time allows us to track the change of C-O symmetric

population over time when we take area difference from the initial area of the symmetric population. However, as shown in Figure 3.4C, the changes in the symmetric and asymmetric carboxylate peak percentages exceed the relative percentage of the free ligand carbonyl peak. This observation implies that the carboxylate species are being converted into other products in addition to free ligands. Examination of higher frequency vibration shows that a peak at 2335 cm^{-1} grows over time, suggesting that this is the byproduct into which the carboxylate is being converted (Figure 3.5A) . This peak is associated with the production of CO_2,[39,66] implying that the oleate ligand undergoes decarboxylation. Once decarboxylation occurs, the bonded carboxylate populations are unable to revert to primarily chelating configurations and remain in the bridging moiety over the course of days as seen in Figure 3.5B . In contrast, simply heating and cooling a $Cd(Ol)_2$ solution without prolonged high-temperature exposure is highly reversible. Thus, heating $Cd(Ol)_2$ to the point of decarboxylation likely leads to the formation of a chemical gel , while shorter heating periods favor the formation of a physical gel.

Decarboxylation of metal carboxylate salts to yield ketones with water and CO_2 as byproducts has been well documented.[14,42,44,45,47] In this case, the remaining oleic acid in the $Cd(Ol)_2$ solution is integral for this decarboxylative formation of a chemical gel as per Equation 3.1.

$$2RCOOH + 2Cd(RCOO^-)_2 \rightarrow CO_2 + R_2C{=}O + [Cd(COO^-)_2]_2 + H_2O \qquad (3.1)$$

More specifically, with octa-9-decene as our R group, pentatriacont-9,26-dien-18-one is produced with CO_2, H_2O, and a dicadmium oleate complex, which necessitates a bridging moiety. While previously we have attributed the 1760 cm^{-1} peak to the oleic acid monomer, it could also be attributed to the produced pentatriacont-9,26-dien-18-one as ketonic products have had vibrations documented between 1720 cm^{-1} and 1780 cm^{-1}.[45–47]Similar decarboxylation reactions have been shown to occur

between 200°C and 300°C on InP magic sized clusters,[12] at 280°C for the production of acetone from calcium acetate,[47] and at 170°C for metal oxide catalyzed decomposition of metal oleates.[44,46] Likewise, there is much documentation that the $Cd(Ol)_2$ chemical gels are discotic in macrostructure with the $[Cd(COO^-)_2]_2$ units stacked on top of each other.[15,19,36,41] Likely, the ketonic byproduct links such layers together. As such, this decarboxylative gel formation happens at a significantly low temperature.

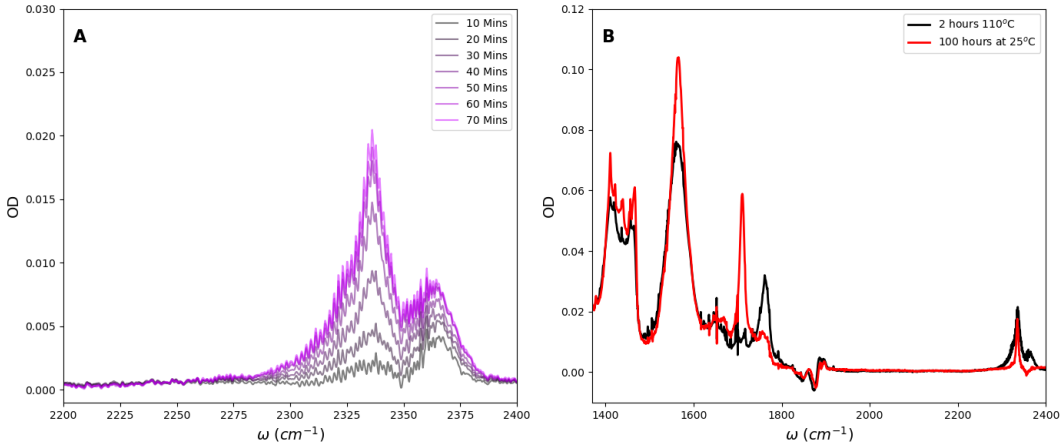

Figure 3.5: Growth of the CO_2 peak at 2335 cm^{-1} over time (A) and a comparison of the FTIR spectra of the chem gel of $Cd(Ol)_2$ after heating for 2 hours at 110°C and 100 hours at room temperature post heating (B).

Although these gel types appear indistinguishable to the eye, their chemical and physical properties can differ significantly due to their different physical makeups seen in Figure 3.6 . The short heated $Cd(Ol)_2$ is simply formed due to the London dispersion forces between the alkyl tails of the oleate. These intermolecular interactions force a physical gel to be supported with solvent molecules. In contrast, the decarboxylation of $Cd(Ol)_2$ in the presence of oleic acid results in the formation of a dicadmium tetraoleate complex that forms a discotic stacked structure scaffolded with the ketonic product from decarboxylation.[54] Understanding the various meth-

ods and mechanisms underlying gel formation is crucial for optimizing gel properties synthetically. Such optimization is particularly important as the viscosity of the gel can influence the synthesis of quantum dot (QD) nanostructures.[15,51] Additionally, by fine-tuning the gel's porosity, it is possible to tailor its properties for applications such as gas adsorption.[18]

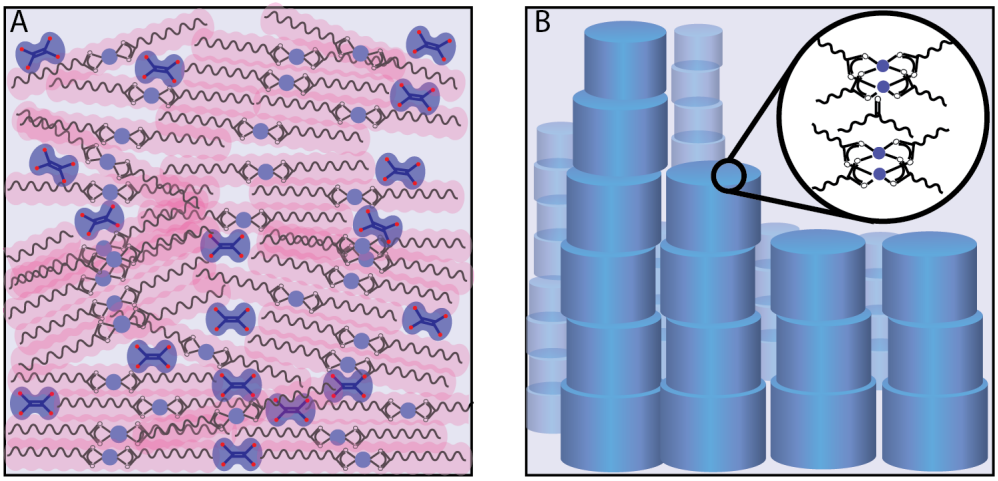

Figure 3.6: The physical gel of $Cd(Ol)_2$ formed via london dispersion forces between alkyl tails (A), and the chemical gel of $[Cd(Ol)_2]_2$ with a bamboo like structure of dicadmium complexes (B).

3.4 Conclusion

In this study, we utilized temperature-dependent FTIR and ^{1}H NMR spectroscopy to examine and compare the surface ligand populations of quantum dot (QD) solutions washed with either MeCN or EtOH during gelation. Our results show that the MeCN-washed QDs contained a significant amount of $Cd(Ol)_2$ impurities, and we confirm that these impurities play a key role in promoting gelation. In contrast, the EtOH-washed CdS QDs primarily exhibited X-type ligands in both chelating and bridging binding modes, as determined through chemical analysis via ICP. Heating studies

of $Cd(Ol)_2$ were conducted to elucidate the gelation dynamics, revealing that short heating times lead to the formation of a physical gel, while extended heating, which induces some irreversible decarboxylation, results in a chemical gel. These different gel types likely exhibit distinct viscoelastic properties, which could be leveraged for specific applications in materials science.

BIBLIOGRAPHY

(1) An, Y.; Solis, F. J.; Jiang, H. *Journal of the Mechanics and Physics of Solids* **2010**, *58*, 2083–2099.

(2) Arachchige, I. U.; Brock, S. L. *Acc. Chem. Res.* **2007**, *40*, 801–809.

(3) Bardelang, D.; Zaman, M. B.; Moudrakovski, I. L.; Pawsey, S.; Margeson, J. C.; Wang, D.; Wu, X.; Ripmeester, J. A.; Ratcliffe, C. I.; Yu, K. *Adv. Mater.* **2008**, *20*, 4517–4520.

(4) Bharti, S. K.; Roy, R. *TrAC Trends in Analytical Chemistry* **2012**, *35*, 5–26.

(5) Calvin, J. J.; O'Brien, E. A.; Sedlak, A. B.; Balan, A. D.; Alivisatos, A. P. *ACS Nano* **2021**, *15*, 1407–1420.

(6) Calvin, J. J.; Swabeck, J. K.; Sedlak, A. B.; Kim, Y.; Jang, E.; Alivisatos, A. P. *J. Am. Chem. Soc.* **2020**, *142*, 18897–18906.

(7) Cass, L. C.; Malicki, M.; Weiss, E. A. *Anal. Chem.* **2013**, *85*, 6974–6979.

(8) Chatterjee, S.; Kuppan, B.; Maitra, U. *Dalton Trans.* **2018**, *47*, 2522–2530.

(9) Chen, Y.; Cordero, J. M.; Wang, H.; Franke, D.; Achorn, O. B.; Freyria, F. S.; Coropceanu, I.; Wei, H.; Chen, O.; Mooney, D. J.; Bawendi, M. G. *Angew. Chem. Int. Ed Engl.* **2018**, *57*, 4652–4656.

(10) Cossairt, B. M. *Chem. Mater.* **2016**, *28*, 7181–7189.

(11) Cotta, M. A. *ACS Applied Nano Materials* **2020**, *3*, 4920–4924.

(12) Cros-Gagneux, A.; Delpech, F.; Nayral, C.; Cornejo, A.; Coppel, Y.; Chaudret, B. *Journal of the American Chemical Society* **2010**, *132*, 18147–18157.

(13) Drijvers, E.; De Roo, J.; Martins, J. C.; Infante, I.; Hens, Z. *Chem. Mater.* **2018**, *30*, 1178–1186.

(14) Dzik, W. I.; Lange, P. P.; Goossen, L. J. *Chemical Science* **2012**, *3*, 2671–2678.

(15) Enomoto, K.; Takeda, K.; Iwata, N.; Adachi, K.; Kikitsu, T.; Ishida, Y.; Hashizume, D.; Tanaka, M.; Kawakami, H.; Pu, Y.-J. *ACS Appl. Nano Mater.* **2022**, *5*, 3756–3762.

(16) Gaponik, N.; Herrmann, A.-K.; Eychmüller, A. *J. Phys. Chem. Lett.* **2012**, *3*, 8–17.

(17) García-Rodríguez, R.; Liu, H. *Chemical Communications* **2013**, *49*, 7857–7859.

(18) Geng, X.; Liu, D.; Hewa-Rahinduwage, C. C.; Brock, S. L.; Luo, L. *Accounts of Chemical Research* **2023**, *56*, 1087–1096.

(19) Giroud-Godquin, A. M.; Marchon, J. C.; Guillon, D.; Skoulios, A. *The Journal of Physical Chemistry* **1986**, *90*, 5502–5503.

(20) Green, M. L. H. *Journal of Organometallic Chemistry* **1995**, *500*, 127–148.

(21) Hamachi, L. S.; Jen-La Plante, I.; Coryell, A. C.; De Roo, J.; Owen, J. S. *Chemistry of Materials* **2017**, *29*, 8711–8719.

(22) Hamachi, L. S.; Yang, H.; Jen-La Plante, I.; Saenz, N.; Qian, K.; Campos, M. P.; Cleveland, G. T.; Rreza, I.; Oza, A.; Walravens, W., et al. *Chemical Science* **2019**, *10*, 6539–6552.

(23) Hartley, C. L.; Dempsey, J. L. *Nano Letters* **2019**, *19*, 1151–1157.

(24) Hewa-Rahinduwage, C. C.; Geng, X.; Silva, K. L.; Niu, X.; Zhang, L.; Brock, S. L.; Luo, L. *J. Am. Chem. Soc.* **2020**, *142*, 12207–12215.

(25) Jang, E.; Jang, H. *Chemical Reviews* **2023**, *123*, 4663–4692.

(26) Kato, T.; Hirai, Y.; Nakaso, S.; Moriyama, M. *Chemical Society Reviews* **2007**, *36*, 1857–1867.

(27) Klein, M. D.; Bisted, C. H.; Dou, F. Y.; Sandwisch, J. W.; Cossairt, B. M.; Khalil, M. *J. Phys. Chem. C* **2023**, *127*, 16970–16978.

(28) Knauf, R. R.; Lennox, J. C.; Dempsey, J. L. *Chemistry of Materials* **2016**, *28*, 4762–4770.

(29) Korala, L.; Wang, Z.; Liu, Y.; Maldonado, S.; Brock, S. L. *Acs Nano* **2013**, *7*, 1215–1223.

(30) Kroupa, D. M.; Voros, M.; Brawand, N. P.; Bronstein, N.; McNichols, B. W.; Castaneda, C. V.; Nozik, A. J.; Sellinger, A.; Galli, G.; Beard, M. C. *The Journal of Physical Chemistry Letters* **2018**, *9*, 3425–3433.

(31) Kumar, P.; Saxena, N.; Singh, F.; Agarwal, A. *Physica B: Condensed Matter* **2012**, *407*, 3347–3351.

(32) Leger, J. D.; Friedfeld, M. R.; Beck, R. A.; Gaynor, J. D.; Petrone, A.; Li, X.; Cossairt, B. M.; Khalil, M. *J. Phys. Chem. Lett.* **2019**, *10*, 1833–1839.

(33) Li, Y.; Young, D. J.; Loh, X. J. *Materials Chemistry Frontiers* **2019**, *3*, 1489–1502.

(34) Lifshitz, E. *J. Phys. Chem. Lett.* **2015**, *6*, 4336–4347.

(35) Liu, C.; Li, Q.; Wang, H.; Wang, G.; Shen, H. *Nanomaterials (Basel)* **2022**, *12*.

(36) Liu, G.; Conn, C. E.; Drummond, C. J. *The Journal of Physical Chemistry B* **2009**, *113*, 15949–15959.

(37) Mirin, R.; Gossard, A.; Bowers, J. *Electronics Letters* **1996**, *32*, 1732–1734.

(38) Moon, H.; Lee, C.; Lee, W.; Kim, J.; Chae, H. *Advanced Materials* **2019**, *31*, 1804294.

(39) Nelander, B. *Chemical Physics Letters* **1976**, *42*, 187–189.

(40) Nepomnyashchii, A. B.; Harris, R. D.; Weiss, E. A. *Analytical chemistry* **2016**, *88*, 3310–3316.

(41) Nevers, D. R.; Williamson, C. B.; Savitzky, B. H.; Hadar, I.; Banin, U.; Kourkoutis, L. F.; Hanrath, T.; Robinson, R. D. *Journal of the American Chemical Society* **2018**, *140*, 3652–3662.

(42) Nicholson, J. W.; Wilson, A. D. *Journal of chemical education* **2004**, *81*, 1362.

(43) Nozik, A. J.; Beard, M. C.; Luther, J. M.; Law, M.; Ellingson, R. J.; Johnson, J. C. *Chemical reviews* **2010**, *110*, 6873–6890.

(44) Park, K.; Lee, S. *RSC advances* **2013**, *3*, 14165–14182.

(45) Pham, T. N.; Sooknoi, T.; Crossley, S. P.; Resasco, D. E. *Acs Catalysis* **2013**, *3*, 2456–2473.

(46) Rajadurai, S. *Catalysis Reviews* **1994**, *36*, 385–403.

(47) Renz, M. *European journal of organic chemistry* **2005**, *2005*, 979–988.

(48) Ritchhart, A.; Cossairt, B. M. *Angew. Chem.* **2018**, *130*, 1926–1930.

(49) Ritchhart, A. Quantitative Analysis and Modification of Colloidal Nanoparticle Surfaces and Structures, Ph.D. Thesis, University of Washington, 2020.

(50) Ritchhart, A.; Cossairt, B. M. *Inorg. Chem.* **2019**, *58*, 2840–2847.

(51) Safaie, B.; Youssefi, M.; Rezaei, B. *Polym. Bull.* **2019**, *76*, 4335–4354.

(52) Sayevich, V.; Cai, B.; Benad, A.; Haubold, D.; Sonntag, L.; Gaponik, N.; Lesnyak, V.; Eychmüller, A. *Angew. Chem. Int. Ed Engl.* **2016**, *55*, 6334–6338.

(53) Shanavas, K. V.; Sharma, S. M.; Dasgupta, I.; Nag, A.; Hazarika, A.; Sarma, D. D. *J. Phys. Chem. C Nanomater. Interfaces* **2012**, *116*, 6507–6511.

(54) Shi, C.; Zhu, J. *Chem. Mater.* **2007**, *19*, 2392–2394.

(55) Sichert, J. A.; Tong, Y.; Mutz, N.; Vollmer, M.; Fischer, S.; Milowska, K. Z.; Garc, R.; Nickel, B.; Cardenas-daw, C.; Stolarczyk, J. K.; Urban, A. S.; Feldmann, J. **2015**.

(56) Sone, E. D.; Zubarev, E. R.; Stupp, S. I. *Angew. Chem. Int. Ed Engl.* **2002**, *41*, 1706–1709.

(57) Talapin, D. V.; Lee, J. S.; Kovalenko, M. V.; Shevchenko, E. V. *Chem. Rev.* **2010**, *110*, 389–458.

(58) Tatebayashi, J.; Kako, S.; Ho, J.; Ota, Y.; Iwamoto, S.; Arakawa, Y. *Nature Photonics* **2015**, *9*, 501–505.

(59) Urban, J. J.; Talapin, D. V.; Shevchenko, E. V.; Kagan, C. R.; Murray, C. B. *Nat. Mater.* **2007**, *6*, 115–121.

(60) Veamatahau, A.; Jiang, B.; Seifert, T.; Makuta, S.; Latham, K.; Kanehara, M.; Teranishi, T.; Tachibana, Y. *Phys. Chem. Chem. Phys.* **2015**, *17*, 2850–2858.

(61) Welsch, T. A.; Cleveland, J. M.; Thomas, J. A.; Schyns, Z. O. G.; Korley, L. T. J.; Doty, M. F. *ACS Appl. Nano Mater.* **2024**, *7*, 13319–13327.

(62) Wu, N.; Fu, L.; Su, M.; Aslam, M.; Wong, K. C.; Dravid, V. P. *Nano Lett.* **2004**, *4*, 383–386.

(63) Xiang, X.; Wang, L.; Zhang, J.; Cheng, B.; Yu, J.; Macyk, W. *Advanced Photonics Research* **2022**, *3*, 2200065.

(64) Yan, J.-J.; Wang, H.; Zhou, Q.-H.; You, Y.-Z. *Macromolecules* **2011**, *44*, 4306–4312.

(65) Yang, Y.; Qin, H.; Jiang, M.; Lin, L.; Fu, T.; Dai, X.; Zhang, Z.; Niu, Y.; Cao, H.; Jin, Y.; Zhao, F.; Peng, X. *Nano Lett.* **2016**, *16*, 2133–2138.

(66) Yin, S.; Zhou, Y.; Liu, Z.; Wang, H.; Zhao, X.; Zhu, Z.; Yan, Y.; Huo, P. *Nature Communications* **2024**, *15*, 437.

(67) Yuan, F.; Yuan, T.; Sui, L.; Wang, Z.; Xi, Z.; Li, Y.; Li, X.; Fan, L.; Tan, Z.; Chen, A., et al. *Nature communications* **2018**, *9*, 2249.

(68) Zeleňák, V.; Vargová, Z.; Györyová, K. *Spectrochimica Acta - Part A: Molecular and Biomolecular Spectroscopy* **2007**, *66*, 262–272.

Appendix A

SUPPLEMENTAL INFORMATION FOR CHAPTER 2

Table A.1: Parameters extracted from an MCMC global fit on a temperature series of FTIR spectra from an InP MSC sample with wider prior widths. Values and errors are given as means and standard deviations among the sampled points in the top 15,000 posterior probabilities, respectively.

Parameter	**Narrow Priors**	**Wide Priors**	**Uninformed**
μ_1	1515.0 ± 2.1	1515.1 ± 3.9	1516.4 ± 5.2
σ_1	10.8 ± 1.0	11.0 ± 2.0	11.3 ± 1.7
μ_2	1537.1 ± 1.3	1537.1 ± 2.5	1537.9 ± 3.4
σ_2	11.2 ± 1.9	12.6 ± 3.2	11.2 ± 1.7
μ_3	1553.9 ± 1.9	1553.4 ± 3.5	1553.3 ± 3.8
σ_3	9.1 ± 0.8	9.3 ± 1.6	10.0 ± 1.8
μ_4	1577.0 ± 0.28	1577.0 ± 0.53	1577.2 ± 2.9
σ_4	12.2 ± 0.27	12.2 ± 0.53	12.4 ± 1.6
μ_5	1609.0 ± 0.28	1609.0 ± 0.54	1609.4 ± 2.7
σ_5	10.0 ± 0.18	10.0 ± 0.34	11.1 ± 2.0
β	0.62 ± 0.10	0.62 ± 0.10	0.63 ± 0.10

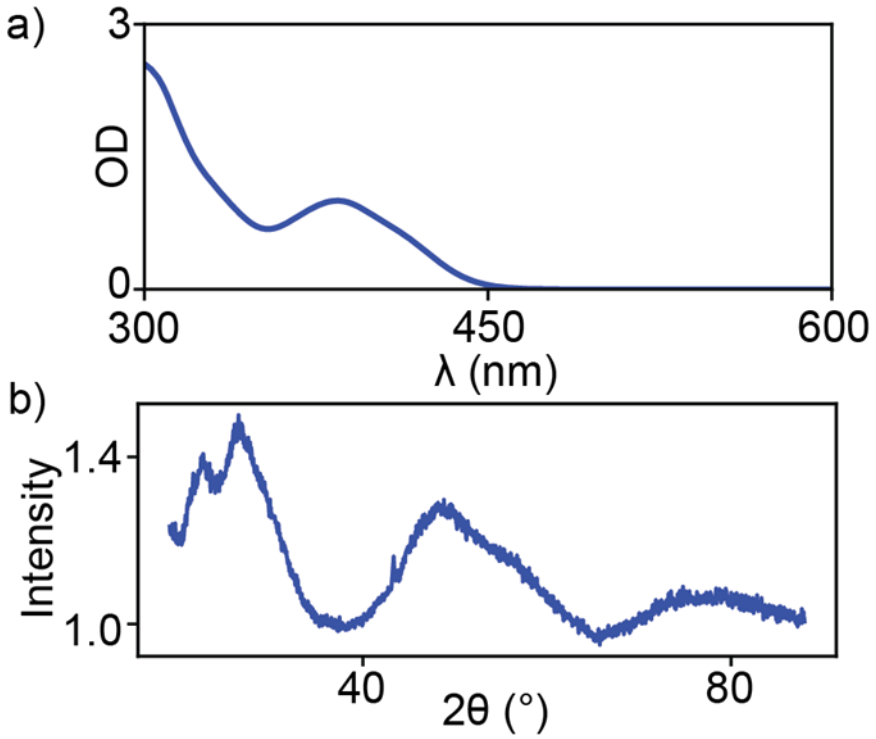

Figure A.1: a) UV-Vis spectrum (top) and X-Ray diffraction trace (bottom) of InP MSC sample

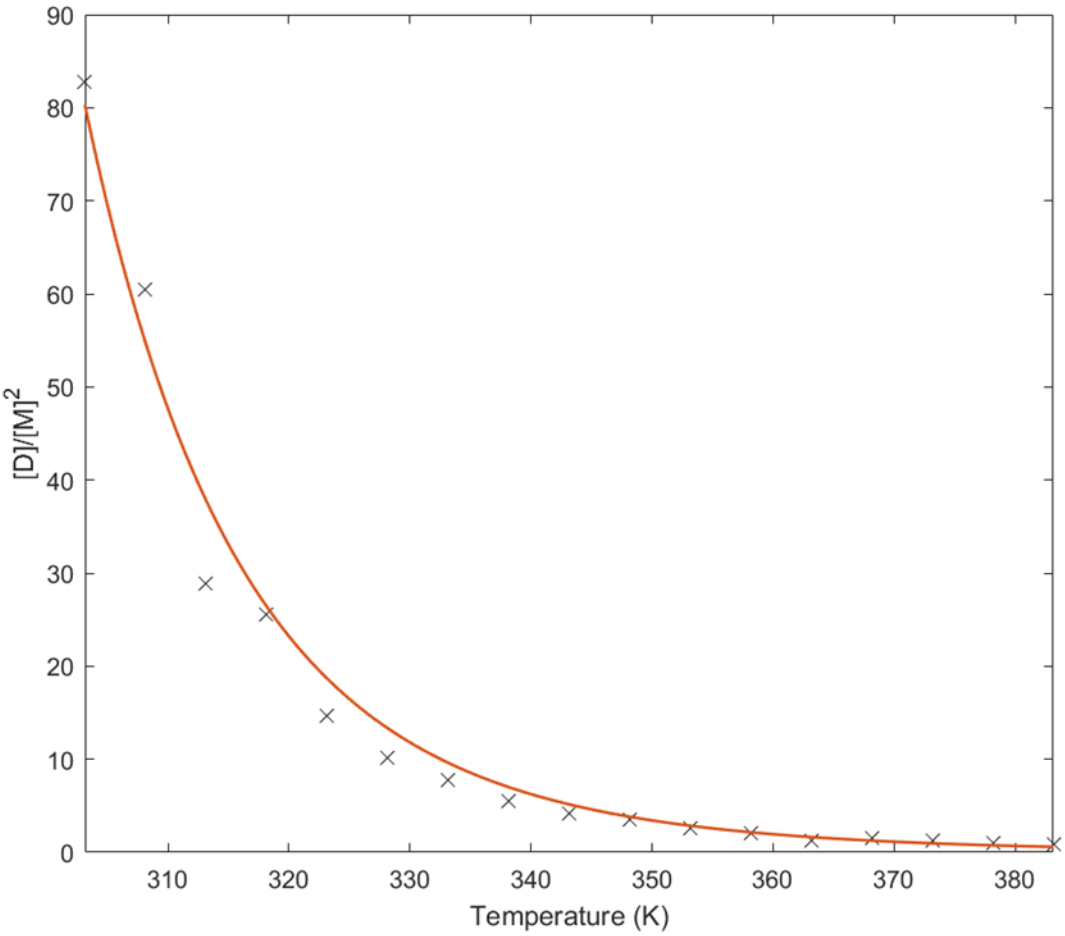

Figure A.2: The ratio of the myristic acid dimer peak areas, $[D]$, over the square of the dilute myristic acid peak area, $[M]^2$, plotted against temperature (black). This data was fit to $\alpha e^{\Delta E/RT}$ corresponding to the temperature dependence of an equilibrium constant, where α is a proportionality constant (orange). The extracted free energy was 3.6 kcal/mol, similar to results reported previously for the dimerization of myristic acid in carbon tetrachloride[1] room temperature data point was excluded from this fit due to a lack of signal from the monomer peak.

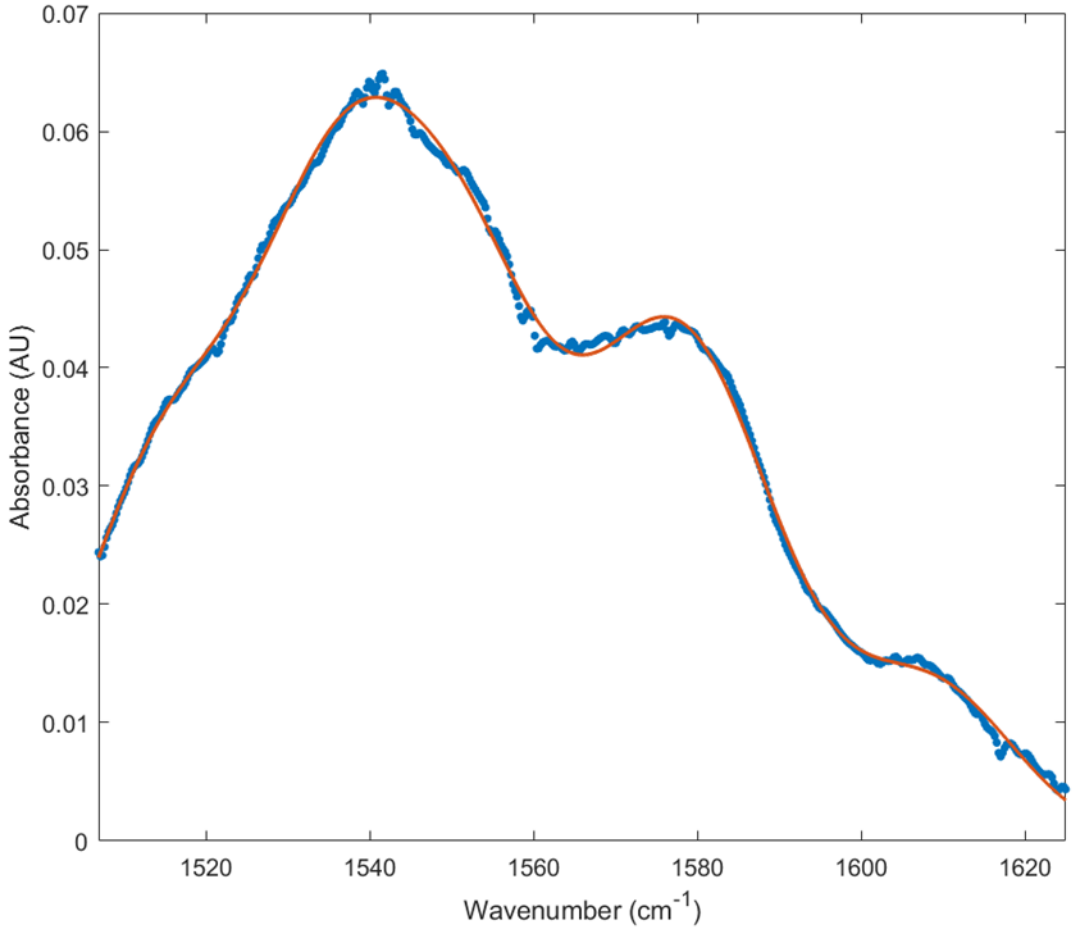

Figure A.3: Room temperature FTIR spectrum of myristic acid-functionalized InP MSCs (blue circles) fit to the sum of five gaussian peaks (orange line). The obtained means and widths for these peaks were $\mu_1 = 1515 \pm 8$, $sigma_1 = 10.8 \pm 3.7$, $\mu_2 = 1537 \pm 5$, $\sigma_2 = 10.8 \pm 7.5$, $\mu_3 = 1554 \pm 8$, $\sigma_3 = 9.1 \pm 3.0$, $\mu_4 = 1577 \pm 1$, $\sigma_4 = 12.2 \pm 1.0$, $\mu_5 = 1609 \pm 1$, $\sigma_5 = 10.0 \pm 0.6$

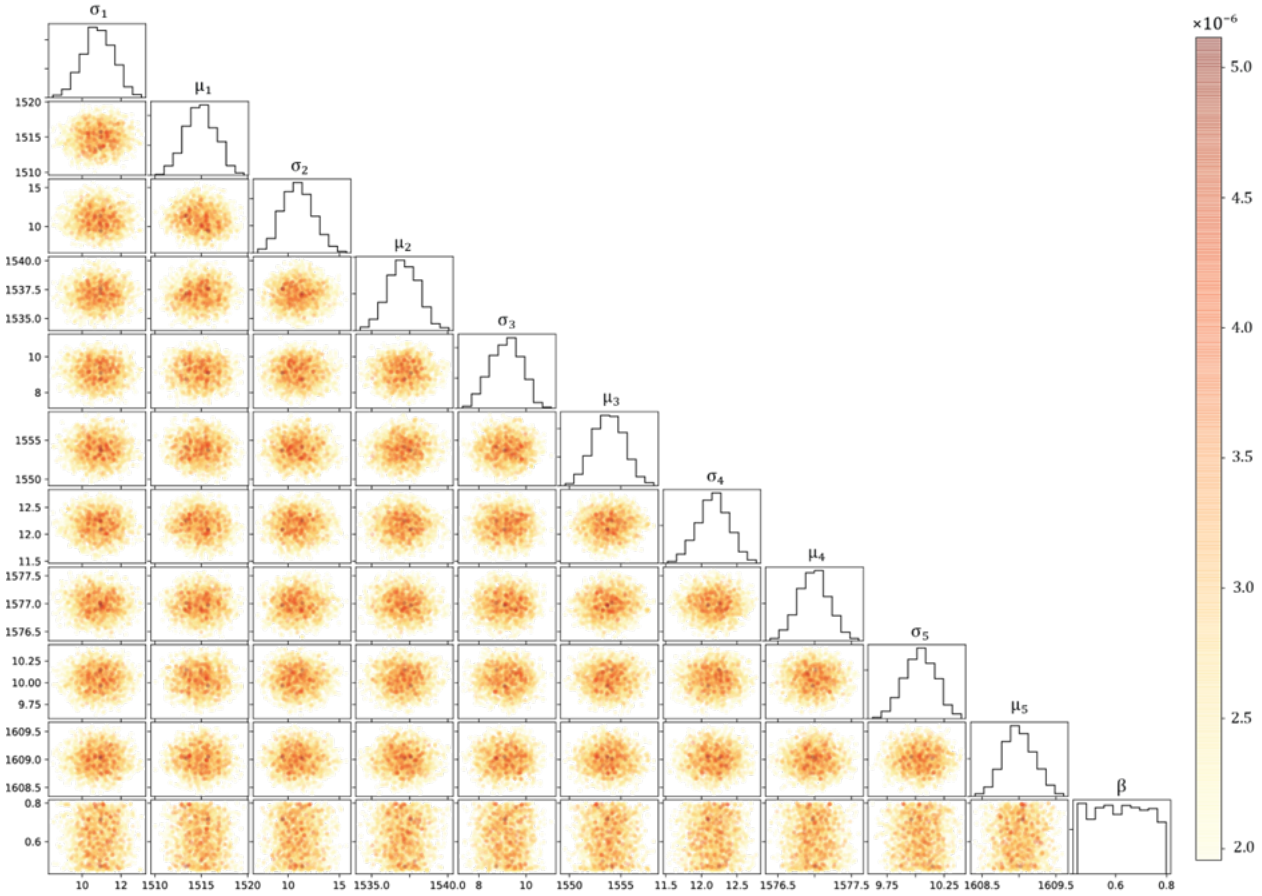

Figure A.4: Correlation plot for top 2000 sampled fits with the highest posterior probabilities from the MCMC fitting method using priors discussed in the text. Colorbar corresponds to the posterior probability of the sample, higher probabilities are shown as opaque red and lower probabilities are shown as transparent yellow. Note that the posterior probabilities values shown in the colorbar are not normalized, as the MCMC approach only cares about maximizing the relative value; the absolute probability itself is difficult to compute and unnecessary.

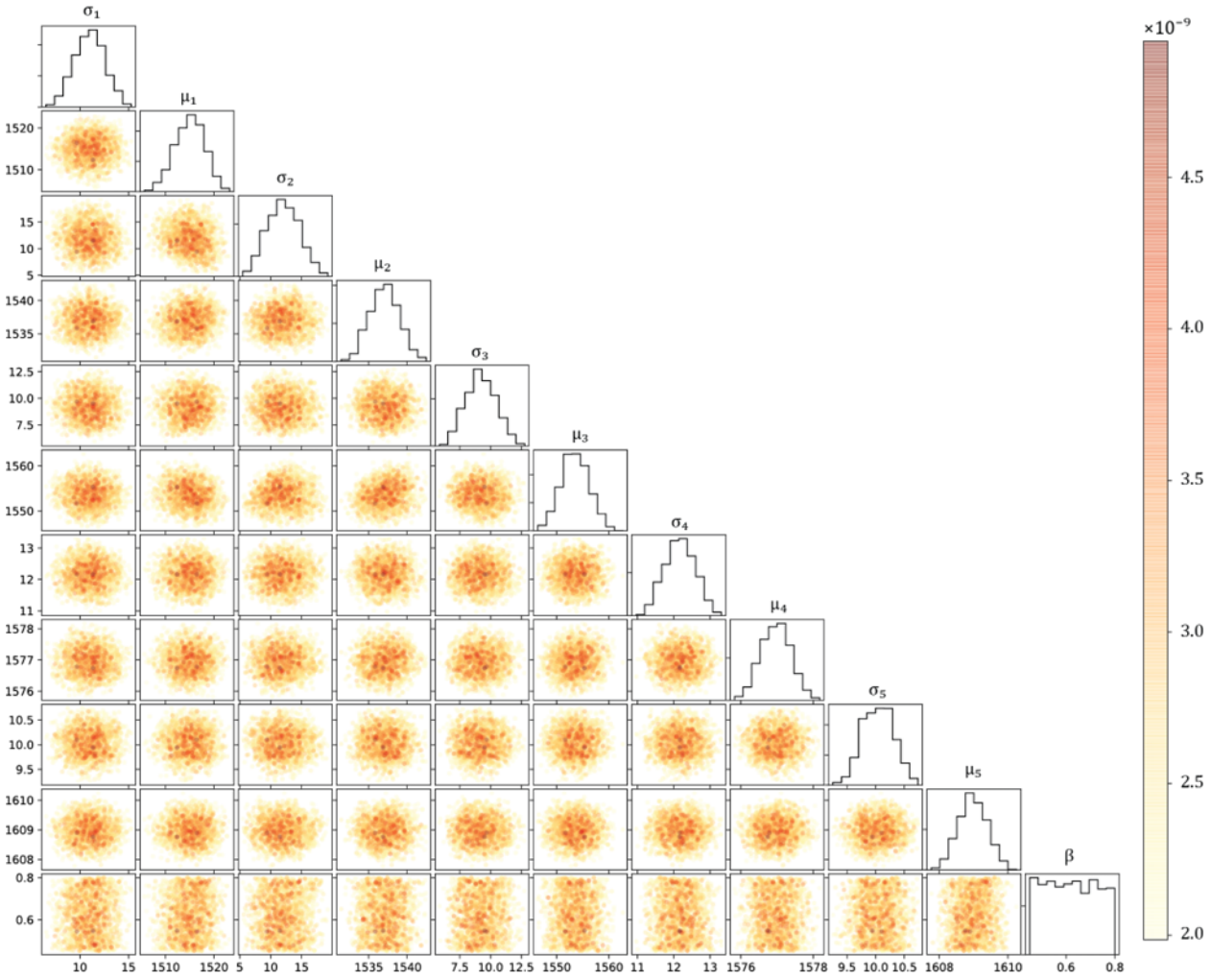

Figure A.5: Correlation plot for MCMC fitting performed using priors with twice as wide of distributions. Colorbar corresponds to the posterior probability of the sample, higher probabilities are shown as opaque red and lower probabilities are shown as transparent yellow. Note that the posterior probabilities values shown in the colorbar are not normalized, as the MCMC approach only cares about maximizing the relative value; the absolute probability is difficult to compute and unnecessary.

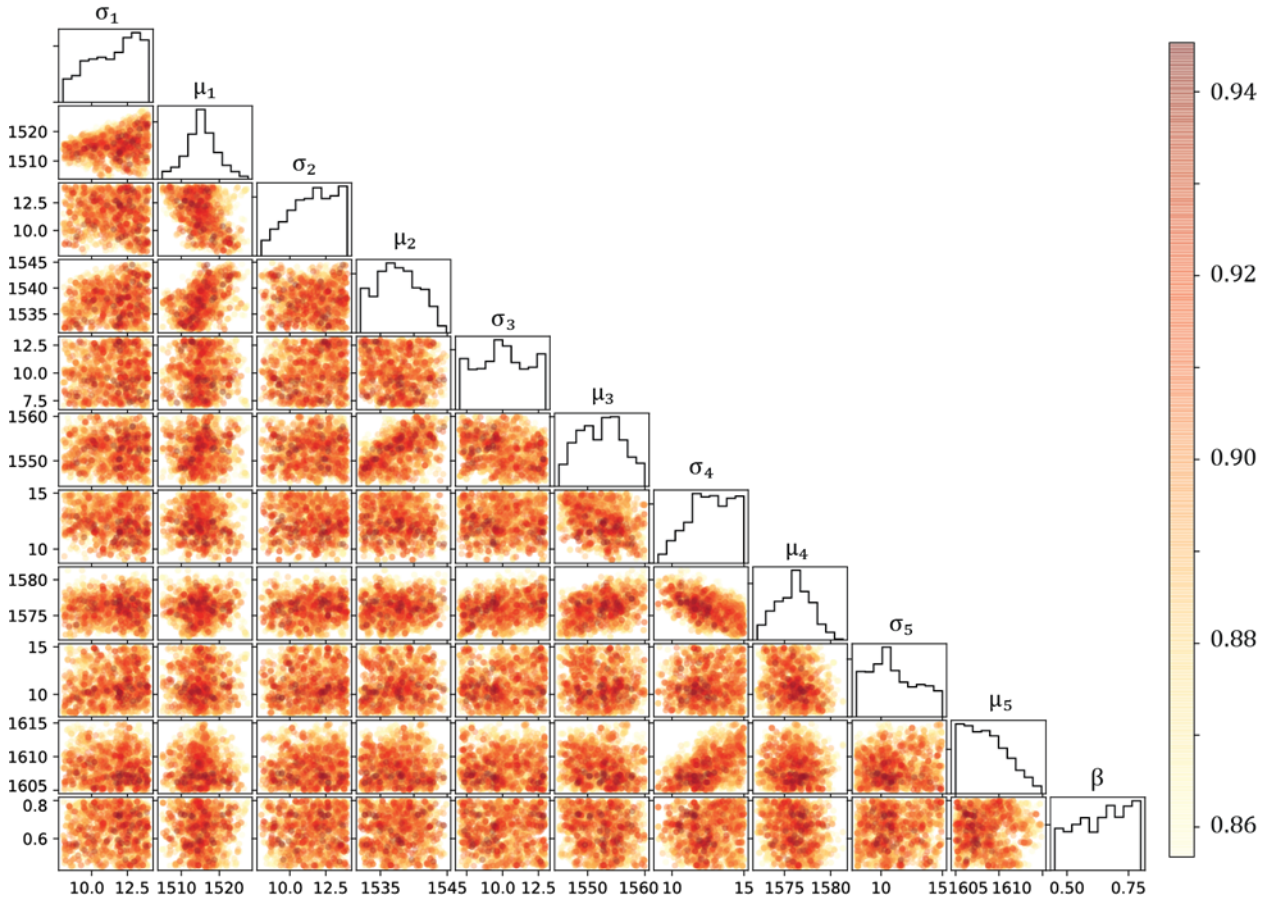

Figure A.6: Correlation plot for 2000 points with highest posterior probabilities from MCMC fitting with uninformed priors. Notably, although the likelihood distribution appears somewhat broad, likely due to the high dimensionality, there is no distinct sign of over- or under-fitting - i.e. variance or bias - respectively. Colorbar corresponds to the posterior probability of the sample, higher probabilities are shown as opaque red and lower probabilities are shown as transparent yellow. As this sampling was done with uninformed priors, the posterior probability is equivalent to the likelihood of the sampled parameters.

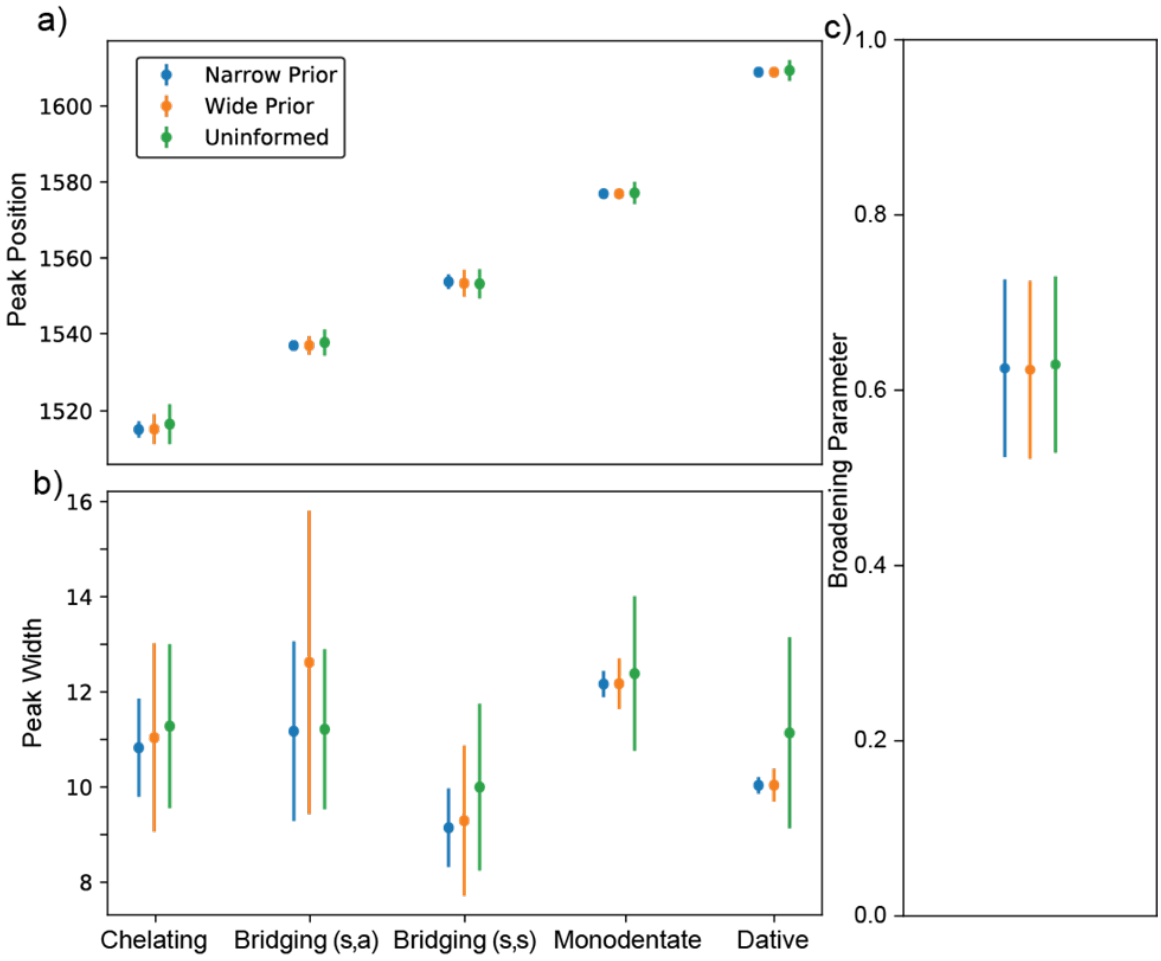

Figure A.7: A comparison of fit parameters and associated errors for the 15,000 sampled points with the highest posterior probabilities given narrow prior distributions, wide priors, or uninformed priors. The peak mean positions are given in (a), the widths in (b), and the broadening parameter in (c).

Table A.2: Relative conformation energies extracted from MCMC global fits on a temperature series of FTIR spectra from an InP MSC sample with varying prior distributions. Values and errors are given as means and standard deviations among the sampled points in the top 15,000 posterior probabilities, respectively. Note that the MCMC with uninformed priors was run for fewer samples than the others (30,000 compared to 300,000), and thus might underestimate the error slightly.

Binding Mode	**Narrow Priors**	**Wide Priors**	**Uninformed**
Chelating	-236 ± 120	-408 ± 734	-217 ± 151
Bridging (s,a)	-35 ± 57	12 ± 136	8 ± 351
Bridging (s,s)	-400 ± 184	-662 ± 908	-484 ± 919
Dative	591 ± 34	607 ± 57	649 ± 177
Free Ligand	1582 ± 19	1594 ± 39	1584 ± 58

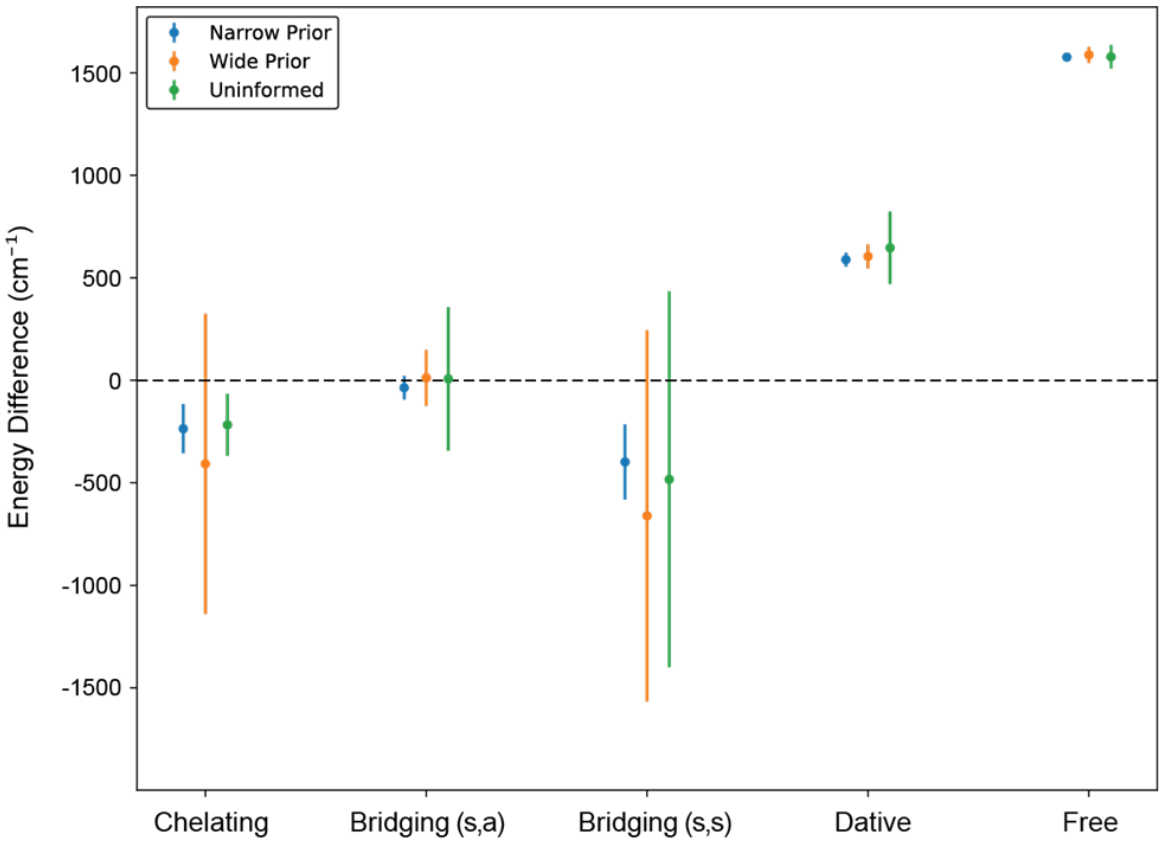

Figure A.8: A comparison of the energy differences obtained via linear regressions of log area differences over the 15,000 fits with the highest posterior probabilities. Results from MCMC run with narrow priors (blue) is compared to wider priors (orange), and uninformed priors (green). Differences are shown as relative energies for binding motifs with respect to the monodentate binding conformation.

Appendix B

SUPPLEMENTAL INFORMATION FOR CHAPTER 3

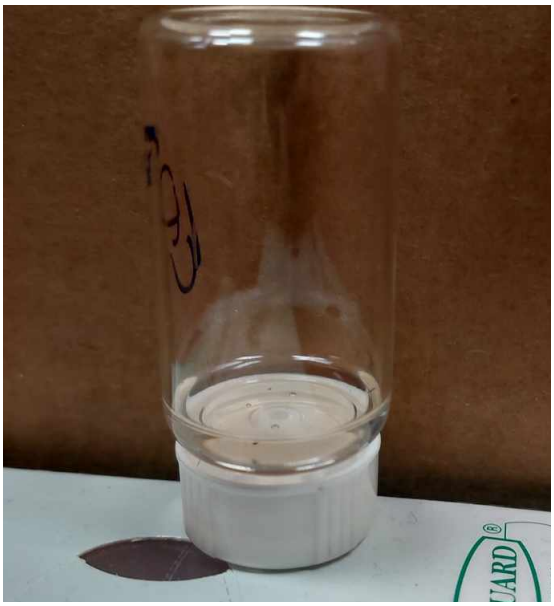

Figure B.1: Unheated MeCN washed CdS-oleate QDs left in ambient conditions for 7 days

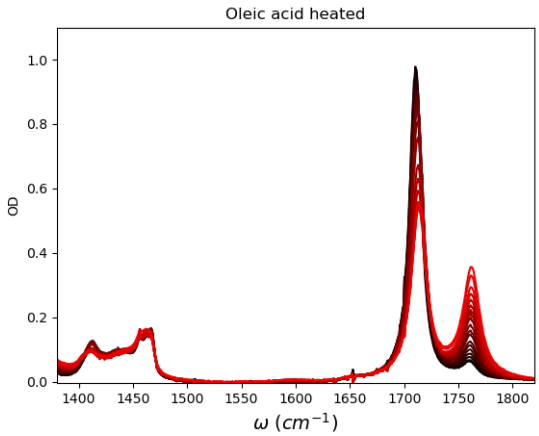

Figure B.2: Oleic Acid heated from 25°C (black) to 110°C (red)

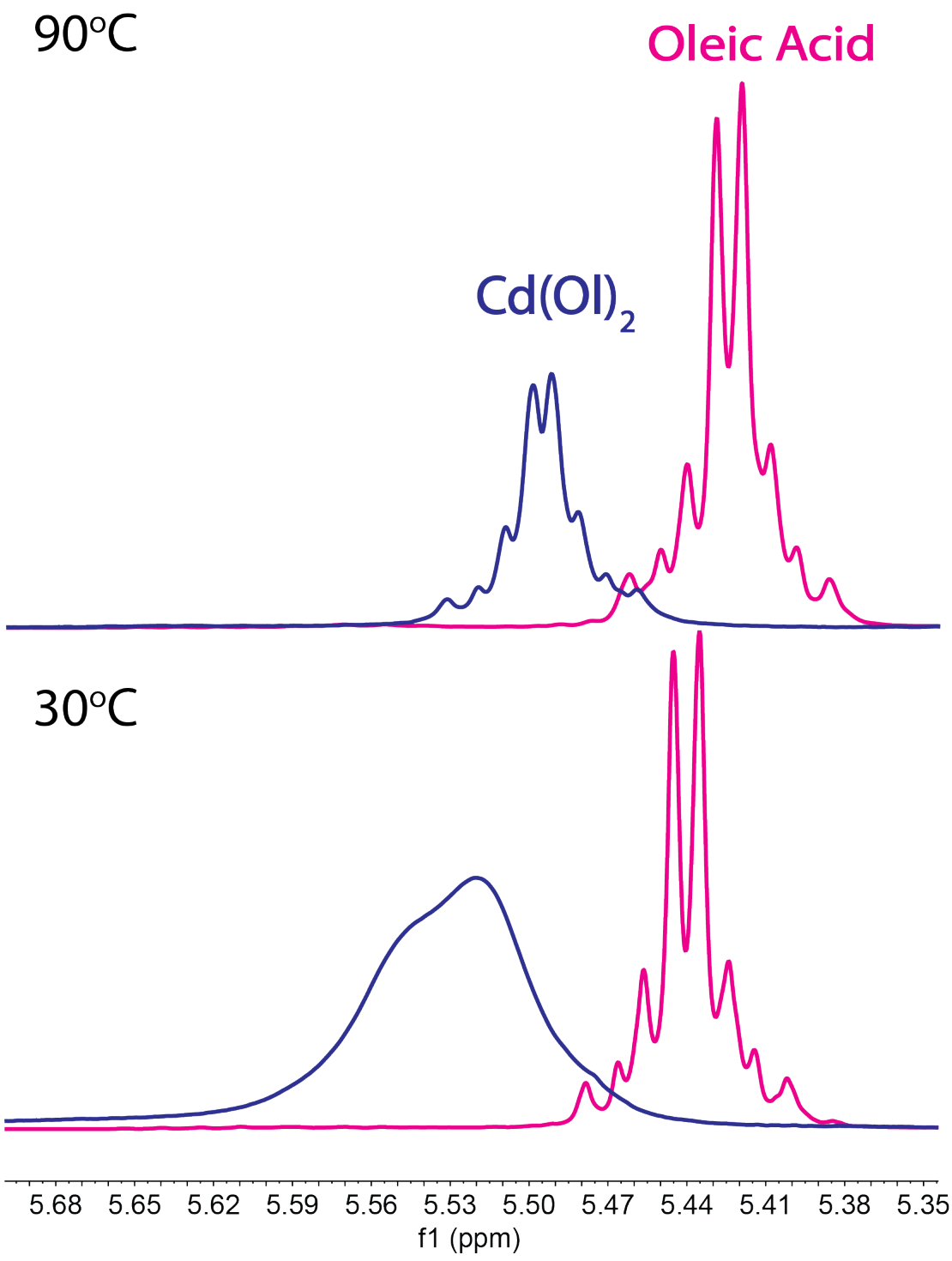

Figure B.3: ^{1}HNMR comparison of $Cd(Ol)_2$ and Oleic Acid at 30^{o}C and 90^{o}C

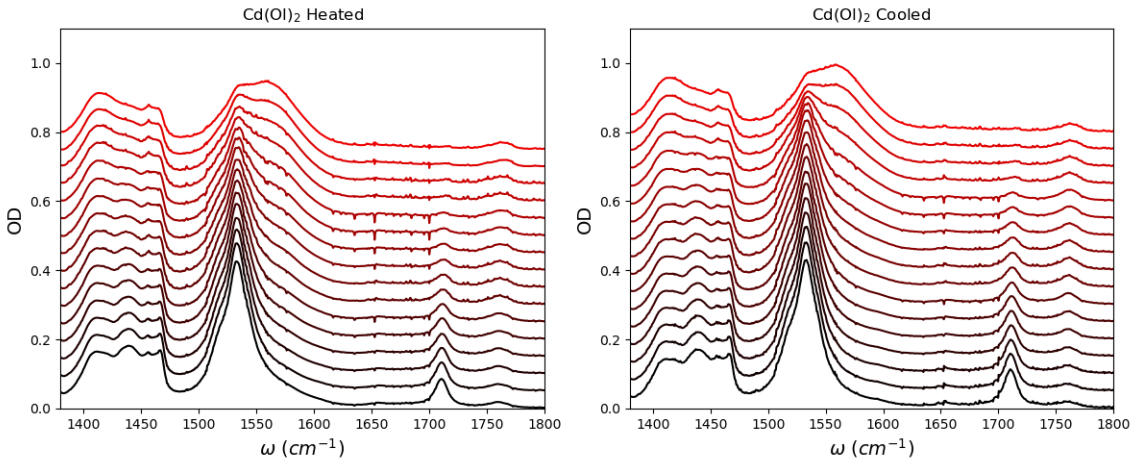

Figure B.4: TD-FTIR spectra from 30^{o}C to 110^{o}C heated and cooled with red spectra being the highest in temperature and black the lowest

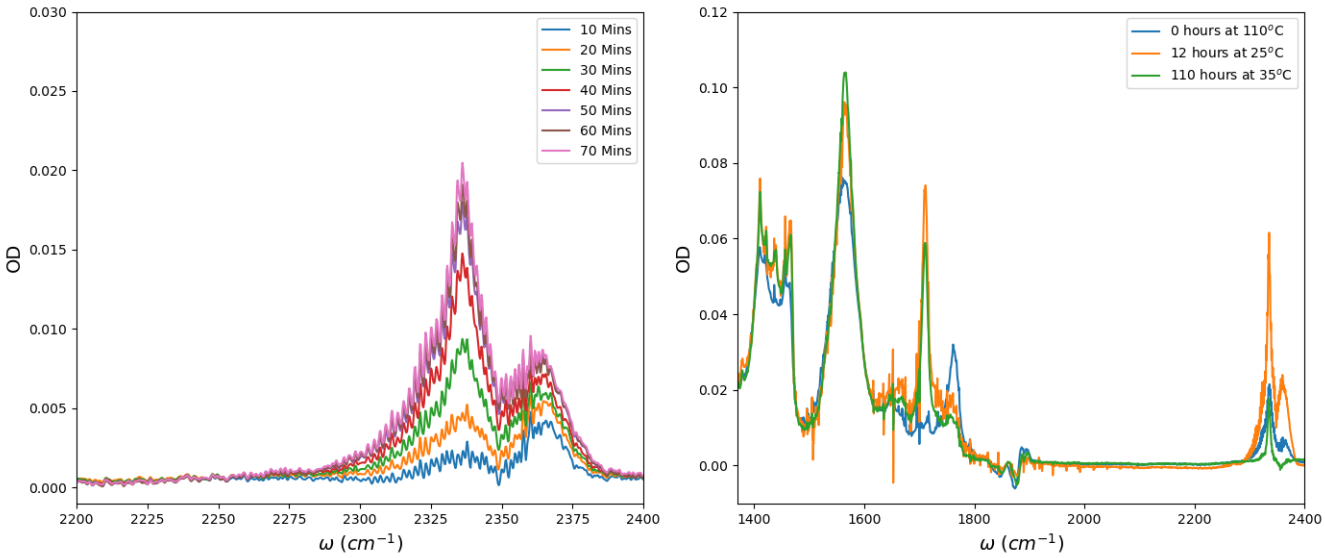

Figure B.5: Generation of CO_2 through decarboxylation of Cd(Ol)$_2$ (left) . This results in a chem gel that does not change over time (right

Appendix C

THE TEMPERATURE DEPENDENT FTIR EXPERIMENT

The general temperature dependent FTIR experiment involves a temperature controlled flow cell connected to a remote control Peltier heater. In the case of all experiments described in this thesis , the temperature control liquid cell is a Harrick TFC-S25- 3 with a Harrick ATK-024-3 Peltier heater. Measurements can be conducted on any FTIR with a demountable cell holder or mount. In the case of the experiments , all FTIR measurements were conducted on a Jasco FTIR 4300 followed by an FTIR 4700.

C.1 Temperature dependent FTIR Set Up

The TFC - S25- 3 should be assembled using the procedure from the TFC -S25-3 manual[1] . For our particular set up, you will need the following ingredients:

- 1 harrick cell and Pelletier heater
- 2 1/8-3/16 inch Hose barbs (Cadence 6511IND) purchased here
- 2 Luer Locking needles at least 4 inches in length
- 2 Teflon Swagelok tube fittings 1/8 inch connector 1/4 tube outer diameter (Swagelok S-200-1-4) purchased here
- Masterflex L/S Precision Pump Tubing (Viton Masterflex 96412-16) purchased here

- Cole Parmer 7553 peristaltic pump
- 2 CaF_2 Windows 25 mm x 2 mm windows Harrick (WFD U25)purchased here

Here are some far too detailed instructions, as this method of set up yields the least amount of trouble upon measurement.

1. Put the first CaF_2 windows into the temperature dependent flow cell
2. Arrange the cut Teflon spacers so that they are on either side of the flow holes of the temperature cell
3. Push the bottom CaF_2 up with a gloved finger with a lens wipe on top, such that it is just above the flow hole. Place the CaF_2 on top of the Teflon spacer covered window, and slowly lower the window sandwich into the temperature cell recess. This set up can be seen in Figure C.1

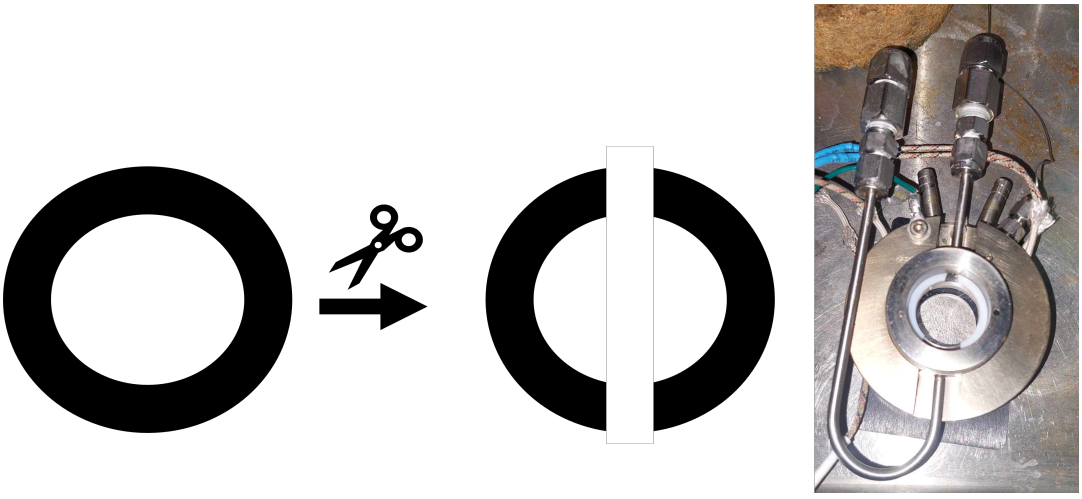

Figure C.1: Spacer and window set up in the temperature control cell

4. Place the black rubber gasket over the top window ensuring that it is not twisted and that it is flush against the side of the recess holding the window

5. Push the stainless steel compression ring as far down over the rubber gasket as possible

6. Screw on the compression nut to finger tightness

7. Screw on the Swagelok tube fittings to the flow cell barbs

8. Cut tubing into one at minimum 11 inch segment and one at minimum 28 inch segment

9. At one end of each tube place a barbed fitting.

10. Place the longer hose segment in the tube fitting attached to the metal J shaped flow inlet and place the shorter hose segment in the other inlet

11. Place the flow cell into the FTIR cell mount, and remove the top of the FTIR (not just the lid).

12. Find a cart, lifter, and ring stand . Place the lifter on top of the cart and raise it such that it is about 4 inches below the opening of the FTIR

13. Place the peristaltic pump on top of the lifter , and readjust the lifters height so the top of the peristaltic pump is equi-height with the height of the FTIR

14. Feed the longer tube segment through the tubing receptacle on the pump ensuring that there is no slack in the tubing when both tubing notches are cinched down and the tubing receptacle tab on the bottom is locked into place (fully to the left as you are looking at the pump)

15. Place needles on Luer locking end of hose fittings

16. Acquire whatever solution of sample you have and adjust the ring stand such that it is slightly above the peristaltic pump, so the needles can be placed into the solution with with out any tension

17. Attach the yellow thermocouple from the temperature cell to the yellow thermocouple receptor on the heater. Plug in the black plug to the ATK heater.

18. Plug in the USB at the back of the ATK heater to the computer

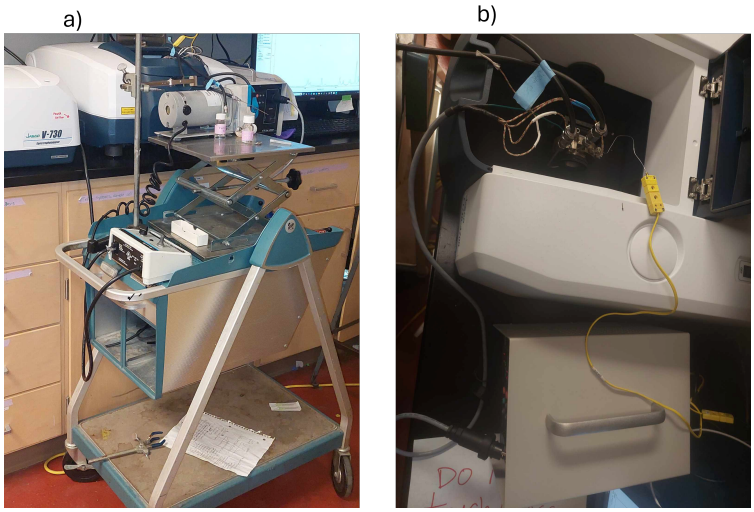

Figure C.2: A) Side view of the experimental showing the peristaltic pump on top of the lifting stage connected to tubing B) The flow cell in the FTIR connected to the ATK heater via grey heating cord and yellow ended wire thermocouple.

C.2 Controlling the Temperature Cell

The temperature cell is controlled by the EZ-Zone configurator software that can be downloaded from Watlow here. Detailed instructions for downloading and setting up the software can be found in the ATK manual.[2] Heating of the temperature cell can be conducted by going to the profile tab, clicking a step , and clicking Ramp Rate

(Figure C.3). The GUI should update and in the "Rate" box you can fill in your desired rate. I do not recommend heating at higher than 3^o/ Min or else you may experience uneven heating or overheating at lower temperatures.You can also hold the cell at a certain temperature using the soak step. For flowing measurements , solutions should be allowed to equilibrate for up to 10 minutes at each temperature. For stationary measurements, solutions were heated at 0.5 ^{o}C per minute . Note: given that a single FTIR measurement takes about 2 minutes, heating does not need to be paused during a measurement unless runaway heating occurs.

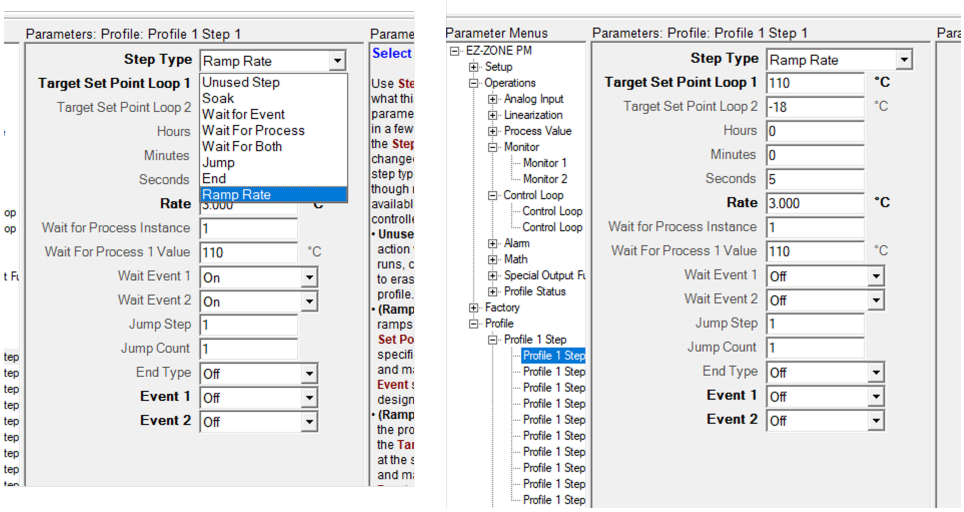

Figure C.3: The EZ-Zone configurator GUI

C.3 FTIR measurements

All measurements in this thesis were averaged 64 times with data taken every 1 cm^{-1}. Background spectra should be taken prior to taking any sample spectra by pressing the blue B button in Figure C.4. Prior to taking a spectra the data acquisition option should be checked , so that .csv and .jws files are both saved with modifiable file names. I suggest that you use the automation script for taking temperature

dependent measurements seen in C.3.2 , so that you do not have to get up and take a measurement every ten minutes.

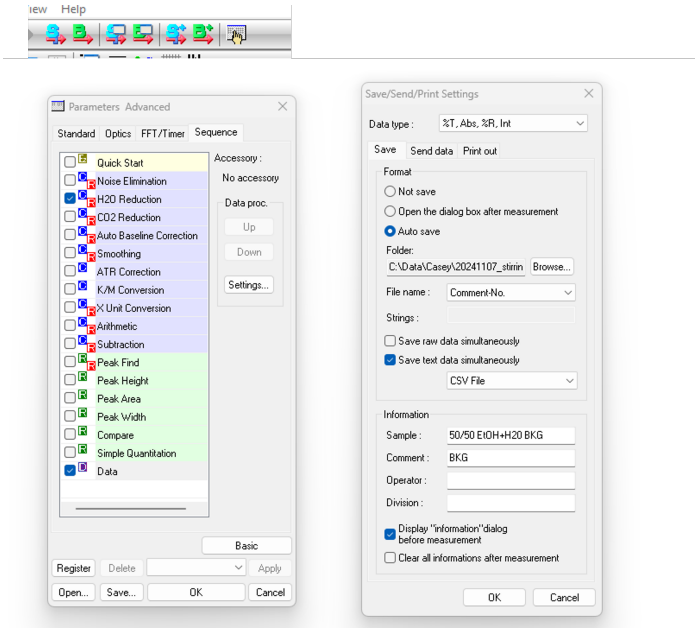

Figure C.4: The Jasco FTIR GUI

C.3.1 Implementation of the Automation Script

The automation script is below in C.3.2.It employs pyautogui for automation of keyboard sequences, Tesseract for image to text conversion, and PIL for showing images. Before even using the automation script, you must set up your computer screen, so that pyautogui can interpret it. I suggest that you split your screen into three parts one with the FTIR spectra acquisition GUI, one with the temperature monitoring screen for the EZ-ZONE GUI, and one for your automation program (Figure C.5). This script relies on using pyautogui to locate the process value active image and the sample spectra image in Figure C.6, whose file paths are labeled process `_value _active _path` and `spectrum_click` respectively. Note: if either of these images are

even one pixel off you will get an image not found error. If this occurs, re - screen shot these images.

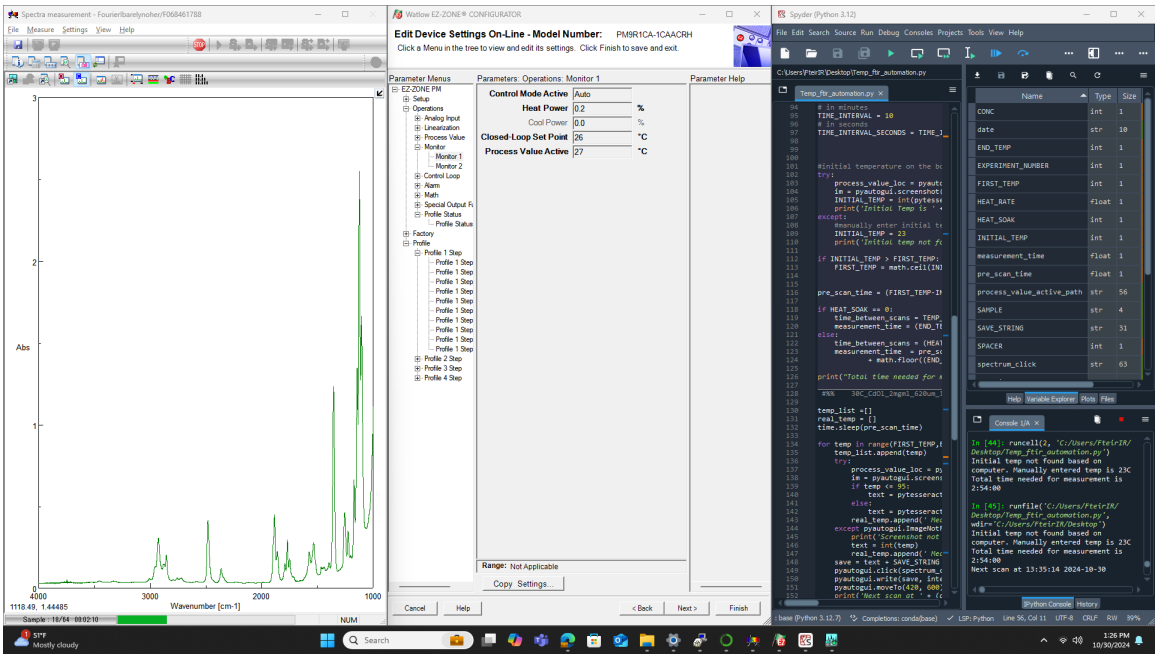

Figure C.5: How to arrange screen windows for the automation process

Figure C.6: Process value and sample spectrum images

C.3.2 The Automation Script

#%% **import** section

import pyautogui

import math

from pytesseract **import** pytesseract

```
from PIL import Image
import time
from datetime import datetime, timedelta
#%% functions

# Convert seconds into hours, minutes, and seconds
def convert(seconds):
    seconds = seconds % (24 * 3600)
    hour = seconds // 3600
    seconds %= 3600
    minutes = seconds // 60
    seconds %= 60
    return "%d:%02d:%02d" % (hour, minutes, seconds)

def  log_results (sample_name:str,
                  spacer_length:int,
                  concentration:int,
                  experiment_number:int,
                  initial_temp :int,
                  temp_interval:int,
                  end_temp:int,
                  save_string ,
                  temp_list ,
                  real_temp_list ):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_filename  = "temperature_log.txt"
    with open(log_filename, "a") as  log_file :
```

```
        log_file.write(f"Log Timestamp: {timestamp}\n")
        log_file.write(f"Sample name: {sample_name}\n")
        log_file.write(f"Spacer length: {spacer_length} micrometers \n")
        log_file.write(f"Concentration: {concentration} mg/ml \n")
        log_file.write(f"Experiment Number: {experiment_number} \n")
        log_file.write(f" Initial  Temperature: {initial_temp} C \n")
        log_file.write(f"Temperature Interval: {temp_interval} seconds\n")
        log_file.write(f"Final Temperature: {end_temp}C\n")
        log_file.write(f"SAVE String: {save_string}\n")
        log_file.write("Temperature List:\n")
        log_file.write(", ".join(map(str, temp_list)) + "\n")
        log_file.write("Real Temperatures List:\n")
        log_file.write(", ".join( real_temp_list ) + "\n")
        log_file.write("\n" + "-"*40 + "\n\n")

#%%  Save file naming and Temperature Detection
tesseract_path = 'C:/Program Files/Tesseract-OCR/tesseract.exe'
process_value_active_path = 'C:/Users/FteirIR/Desktop/Temperatures_automation
    /PVA.png'
spectrum_click = 'C:/Users/FteirIR/Desktop/Temperatures_automation/spec_start.
    png'

startime = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
date = datetime.now().strftime("%Y-%m-%d")
SAMPLE = 'CdOl'
SPACER = 620
CONC = 2
```

```
EXPERIMENT_NUMBER = 3

# degrees Celsius changed per second
HEAT_RATE = 3/60
# Amount of seconds for each heat soak step
HEAT_SOAK = 0
#HEAT_SOAK = 0
# Temp interval to take data in seconds . I recommend not using anything shorter
    than
# 3 minutes since a typical scan takes about 2 and a half minutes
TEMP_INTERVAL = 20

#First temp to measure
FIRST_TEMP = 25
#End temperature in Celsius
END_TEMP = 110


SAVE_STRING = 'C_'+ str(SAMPLE) +'_' + str(CONC) +'mgml_' + str(
    SPACER)+ 'um_' + str(EXPERIMENT_NUMBER)+"_"+date

TIME_MEASUREMENT = True

# in minutes
TOTAL_TIME = 12*60
# in minutes
TIME_INTERVAL = 10
```

```
# in seconds
TIME_INTERVAL_SECONDS = TIME_INTERVAL * 60



#initial temperature on the box in Celsius
try:
    process_value_loc = pyautogui.locateOnScreen(process_value_active_path)
    im = pyautogui.screenshot(region=(process_value_loc[0], process_value_loc [1],
        175, process_value_loc [3]) )
    INITIAL_TEMP = int(pytesseract.image_to_string(im)[-4:-2])
    print(' Initial Temp is ' + str(INITIAL_TEMP) + 'C')
except:
    #manually enter initial temp (red number) on the box
    INITIAL_TEMP = 21
    print(' Initial temp not found based on computer. Manually entered temp is '
        + str(INITIAL_TEMP) + 'C')

if INITIAL_TEMP > FIRST_TEMP:
    FIRST_TEMP = math.ceil(INITIAL_TEMP/TEMP_INTERVAL)*
        TEMP_INTERVAL

pre_scan_time = (FIRST_TEMP-INITIAL_TEMP)/HEAT_RATE

if HEAT_SOAK == 0:
    time_between_scans = TEMP_INTERVAL/HEAT_RATE
    measurement_time = (END_TEMP - INITIAL_TEMP)/HEAT_RATE
```

```
else:
    time_between_scans = (HEAT_SOAK + TEMP_INTERVAL/HEAT_RATE)
    measurement_time = pre_scan_time + HEAT_SOAK \
            + math.floor((END_TEMP - INITIAL_TEMP)/TEMP_INTERVAL) *
                time_between_scans

print("Total time needed for measurement is " + convert(measurement_time))

#%% Temperature experiment Execution
temp_list =[]
real_temp = []
time.sleep(pre_scan_time)

for temp in range(FIRST_TEMP,END_TEMP,TEMP_INTERVAL):
    temp_list.append(temp)
    try:
        process_value_loc = pyautogui.locateOnScreen(process_value_active_path)
        im = pyautogui.screenshot(region=(process_value_loc[0], process_value_loc
            [1], 175, process_value_loc [3]) )
        if temp <= 95:
            text = pytesseract.image_to_string(im)[-4:-2]
        else:
            text = pytesseract.image_to_string(im)[-5:-2]
        real_temp.append(' Measured temp is ' + str(text) + 'C')
    except pyautogui.ImageNotFoundException:
        print('Screenshot not found for ' + str(temp) + 'C' )
        text = str(temp)
```

```
            real_temp.append(' Measured temp screen shot not found for' + str(text) +
                'C')
        save = str(temp) + SAVE_STRING
        pyautogui.click(spectrum_click, interval=0.25);pyautogui.press('tab', interval
            = 0.1);pyautogui.press('back', interval= 0.1);\
        pyautogui.write(save, interval= 0.1);pyautogui.press('Enter')
        pyautogui.moveTo(420, 600)
        print('Next scan at ' + (datetime.now()+timedelta(seconds=
            time_between_scans)).strftime("%H:%M:%S %Y-%m-%d"))
        time.sleep(time_between_scans)
#%% Time experiment Execution

time_list =[]
temp_list_time = []
if TIME_MEASUREMENT:
    for timemin in range(70,TOTAL_TIME+TIME_INTERVAL,
        TIME_INTERVAL):
        time_list .append(timemin)
        try:
            process_value_loc = pyautogui.locateOnScreen(
                process_value_active_path)
            im = pyautogui.screenshot(region=(process_value_loc[0],
                process_value_loc [1], 175, process_value_loc [3]) )
            if int(text) < 95:
                text = pytesseract.image_to_string(im)[-4:-2]
            else:
                text = pytesseract.image_to_string(im)[-5:-2]
```

```
            real_temp.append(' Measured temp is ' + str(text) + 'C')
            temp_list_time.append(' Measured temp is ' + str(text) + 'C' + 'fpr'
                + str(timemin) +'mins')
        except pyautogui.ImageNotFoundException:
            print('Screenshot not found for  ' + str(temp) + 'C' )
            text  = str(temp)
            real_temp.append(' Measured teamp screen shot not found for' + str(
                text) + 'C')
            temp_list_time.append(' Measured temp screen shot not found for' +
                str(timemin) +'mins')
        save =  text+'C_' + str(timemin) + 'mins' + SAVE_STRING
        pyautogui.click(spectrum_click, interval=0.25);pyautogui.press('tab',
            interval= 0.1);pyautogui.press('back',  interval= 0.1);\
        pyautogui.write(save,  interval= 0.1);pyautogui.press('Enter')
        pyautogui.moveTo(420, 600)
        print('Next scan at ' + (datetime.now()+timedelta(minutes=
            TIME_INTERVAL)).strftime("%H:%M:%S %Y-%m-%d"))
        time.sleep(TIME_INTERVAL_SECONDS)



log_results (sample_name = SAMPLE,
                spacer_length  = SPACER,
                concentration=CONC,
                experiment_number= EXPERIMENT_NUMBER,
                initial_temp  = INITIAL_TEMP,
                temp_interval  = TEMP_INTERVAL,
```

```
                end_temp = END_TEMP,
                save_string  = SAVE_STRING,
                temp_list  = temp_list,
                real_temp_list  = real_temp)

print("Temperature data logged successfully.")
#%%
```

C.3.3 Explanation of Automation Code

Below are explanations of what each section of the code does . I suggest running the code section by section, so that if pyautogui isn't locating images you will know immediately. I suggest starting the heating after you run section 3, which determines the initial temperature, but before you run section 4 of the code, which takes spectra every designated amount of minutes. I explain the functionality of each section below

Section 1: Import Statements

Imports the following python packages

- `numpy`
- `pyautogui`
- `tesseract`
- `PIL` (Python Imaging Library)

Section 2: Functions

Defines the functions convert and log_results

- `convert()`
  - Calculates days, hours, and minutes from a given value in seconds.
- `log_results()`
  - Logs the results of the experiment into a specified log file.

Section 3: Save file naming & Temperature Detection

- **Input Variables** Input variables of this script that likely need to be changed every run are below. These will go into a save string for easy experiment identification.
  - `SAMPLE` - Sample ID
  - `SPACER` - Spacer thickness in μm
  - `CONC` - Concentration
  - `EXPERIMENT_NUMBER` - Experiment number
  - `HEAT_RATE` - Rate of heating in degrees Celsius per second
  - `HEAT_SOAK` - Amount of seconds held at each soak step
  - `TEMP_INTERVAL` - °C between each measurement
  - `FIRST_TEMP` - First temperature to be measured
  - `END_TEMP` - Last temperature to be measured
  - `TIME_MEASUREMENT` - Whether the script will take spectra at a specific time increment
  - `TOTAL_TIME` - Total amount of time for an experiment
  - `TIME_INTERVAL` - Amount of time between each measurement
  - `INITIAL_TEMP` - The measured temperature of the sample

- **Process Value Image Detection and Updating Initial Temperature**

  The script attempts to locate the `process_value` image on the screen and captures a screenshot of the area adjacent to it, which shows the active temperature. It then uses `tesseract` OCR (Optical Character Recognition) to extract text from the screenshot, identifying the initial temperature of the temperature cell. If no image is found or no text is extracted, the initial temperature defaults to the input variable `INITIAL_TEMP`.

- **Calculation of necessary instrument times** The script calculates `prescan_time`, which represents the time before the first measurement, using the equation:

$$\texttt{prescan\_time} = \frac{\texttt{FIRST\_TEMP} - \texttt{INITIAL\_TEMP}}{\texttt{HEAT\_RATE}} \tag{C.1}$$

  If `HEAT_SOAK` equals 0, the time between scans and the total measurement time are calculated as:

$$\texttt{time\_between\_scans} = \frac{\texttt{TEMP\_INTERVAL}}{\texttt{HEAT\_RATE}} \tag{C.2}$$

$$\texttt{measurement\_time} = \frac{\texttt{END\_TEMP} - \texttt{INITIAL\_TEMP}}{\texttt{HEAT\_RATE}} \tag{C.3}$$

  If `HEAT_SOAK` is not 0, the time between scans and the total measurement time are calculated using:

$$\texttt{time\_between\_scans} = \texttt{HEAT\_SOAK} + \frac{\texttt{TEMP\_INTERVAL}}{\texttt{HEAT\_RATE}} \tag{C.4}$$

$$\begin{aligned}\texttt{measurement\_time} = \texttt{pre\_scan\_time} + \texttt{HEAT\_SOAK} \\ + \left(\lfloor \frac{\texttt{END\_TEMP} - \texttt{INITIAL\_TEMP}}{\texttt{TEMP\_INTERVAL}} \rfloor \times \texttt{time\_between\_scans} \right)\end{aligned} \tag{C.5}$$

Section 4: Temperature Experiment Execution

The program waits for the calculated time interval, `prescan_time`,which corresponds to the time between the initial temperature and the first measurement. After this waiting period, the program enters a loop that iterates from `FIRST_TEMP` to `END_TEMP` in increments of `TEMP_INTERVAL`. During each iteration of the loop, the following actions are performed: `pyautogui` captures a screenshot of the temperature from the process value location, and `pytesseract` extracts the temperature data. `pyautogui` clicks the "Sample Spectrum" key to initiate a spectrum reading, and `pyautogui` autofills the "Comment" field with a prefix that includes the current temperature and a suffix derived from `save_string`. The measured temperature is recorded in the list `real_temp`, and a message indicating when the next scan will be performed is printed, based on `time_between_scans`. Finally, the program waits for `time_between_scans` before proceeding to the next iteration.

Section 5: Time Experiment Execution

After the waiting period, the program begins a loop that iterates from `FIRST_TIME` to `TOTAL_TIME` in increments of `TIME_INTERVAL`. During each iteration, several actions are performed: the same `pyautogui` key series of filling in save file names is performed. The measured temperature is stored in the list `temp_list_time`, and the corresponding measured time is added to `time_list`. Additionally, a message indicating when the next scan will occur is printed, based on `time_between_scans`, and the program waits for `time_between_scans` before proceeding to the next iteration. Key information is also logged to the log file using `log_results`.

C.4 Importing and Solvent Subtracting the Temperature Dependent Data

I use the following scripts to load and plot temperature data, which can be seen in. I recommend you save all of your data for a given experiment in a specific file as a complex CSV from the FTIR, with a specific name scheme. By this I mean name the 25^{o}C measurement something like 25C_samplename __sampleconcentration.csv and 100C_samplename _sampleconcentration.csv.

1. `rename_and_order` - Takes the inputs file_path and file_string, imports all csv file names from a specific folder. Splits the name at the string and at the backslash, so that temperatures are put in a list called. Then orders these temperatures for the `tempordered` list, and orders the file names in this order in a list `filesordered`. The function returns `tempordered` and `filesordered`.

2. `load_freq_temp_array` - Takes the inputs filesordered, which is a list of strings , and for loops through all strings to load data into a list . This data is converted into an array where each row is a different temperature spectra called `freq_temp _array` . The function returns `freq_temp _array` and the x - axis of the data in wavenumbers.

3. `solvent_sub_arrays_in` Takes the following inputs :

   - `sample_data_array` the sample data array with temperature increasing down rows
   - `solvent_data_array` the solvent data array with temperature increasing down rows. Note: the solvent should have the same temperatures as the measured samples for this function to work
   - `xaxis` the xaxis for the sample and the solvent data

- `solvent_fit_LB` the lower bounds for a portion of the spectra that only contains solvent signal
- `solvent_fit_UB` the upper bounds for a portion of the spectra that only contains solvent signal

With `solvent_fit_LB` and `solvent_fit_UB` , the `xaxis`, `sample_data_array`, and `solvent_data_array` are cut by those bounds. Then the cut data is fit to the equation

$$y = \texttt{solvent\_amp} \times \texttt{solvent} + \texttt{offset} \tag{C.6}$$

by looping through all rows in `sample_data_array` and `solvent_data_array`. Fit params are saved in the `fit_params` variable. Then, each of the `fit_params` are plugged back into the solvent subtraction equation (C.6) and subtracted from each sample. The function returns a list of solvent subtracted spectra.

4. `ezcolorplot` plots the solvent subtracted temperature spectra, and is a function of the following

   - `solventsubbedarray` solvent subbed data array
   - `title` The title of the plotted graph.
   - `buffer` The buffer between spectra in the y direction
   - `xaxis` the xaxis for the sample and the solvent data
   - `instrumentlab` whether or not data was taken in the instrument lab
   - `xlower` the lower bounds for the x - axis for the graph shown. Default 1300 cm $^{-1}$.
   - `xupper` the uppper bounds for the x - axis for the graph shown.Default 1800 cm $^{-1}$.

- `ylower` the lower bounds for the y - axis for the graph shown. Default 0.
- `yupper` the uppper bounds for the y - axis for the graph shown.Default 1.

C.5 Importing and Plotting Scripts

`rename_and_order`

```
import glob
import numpy as np
def rename_and_order(filepath, filestring ):
    files = glob.glob(filepath+'*.csv')
    ogtemp = []
    for file in files :
        firstsplit =file . split ( filestring )
        firstsplit = firstsplit [0]
        finaltempname=firstsplit. rsplit ('\\',1)
        finaltempname=finaltempname[1]
        ogtemp = ogtemp + [int(finaltempname)]
    indices= np.argsort(ogtemp)
    tempordered = [ogtemp[ind] for ind in indices]
    filesordered = [ files [ind] for ind in indices ]
    #print(tempordered)
    return filesordered, tempordered
```

`load_freq_temp_array`

```
def load_freq_temp_array( filesordered ):
    freq_temp_list =[]
    for file in filesordered :
        data=np.loadtxt(open(file, "rb"), delimiter=",", skiprows=19,max_rows
            =12466-19)
```

```
        freq_temp_list = freq_temp_list + [data[:,1]]

    freq_temp_array = np.vstack(freq_temp_list)
    #freq_temp_array = np.transpose(freq_temp_array)
    return freq_temp_array,data[:,0]
```

`solvent_sub_arrays_in`

```
    def solvent_sub_arrays_in(sample_data_array, solvent_data_array,xaxis,
        solvent_fit_LB, solvent_fit_UB):
    x_window_solvent=np.logical_and(xaxis<solvent_fit_UB, xaxis>solvent_fit_LB)

    #cutting our data
    xaxis_cut = xaxis[x_window_solvent]
    solvent_data_array_cut = solvent_data_array[:, x_window_solvent]
    sample_data_array_cut= sample_data_array[:,x_window_solvent]
    fit_params = []
    solvent_subtracted = []
    for ii, solvent in enumerate(solvent_data_array_cut):
        def solvent_sub(f,solv_amp,offset):

            y=solv_amp*solvent+offset

            return y
        params, cov = curve_fit(solvent_sub, xaxis_cut, sample_data_array_cut[ii
            ,:])
        fit_params.append(params)
    for ii, solvent in enumerate(solvent_data_array):
        solvent_subbed = sample_data_array[ii] - solvent_sub(1,fit_params[ii][0],
```

```
        fit_params[ ii ][1])
    #solvent_subbed = solvent_sub(1,Cds_2nm_ss_params[ii][0],
        Cds_2nm_ss_params[ii][1])
    solvent_subtracted.append(solvent_subbed)
return solvent_subtracted
```

`ezcolorplot`

```
def ezcolorplot(solventsubbedarray, title, buffer, instrumentlab = False, xlower
    =1300,xupper=1800,ylower = 0,yupper = 1.0):
    plt.figure( figsize  = (5,4))
    plt.xlim([xlower,xupper])
    plt.ylim([ylower,yupper ])
    plt.xlabel('$\omega$' + ' $(cm^{-1})$', fontsize = 14)
    plt.ylabel('OD', fontsize  = 14)
    plt. title ( title )
    for ii, temp in enumerate(solventsubbedarray):
        RGBcolor = colors.hsv_to_rgb((1, 1, ii/len(solventsubbedarray)))
        if instrumentlab:
            plt.plot(xaxis_instrument, solventsubbedarray[ii] + ii*buffer, color
                =RGBcolor )
        else:
            plt.plot(xaxis, solventsubbedarray[ii] + ii*buffer, color =RGBcolor
                )
    plt.tight_layout ()
    return
```

C.6 Trouble Shooting

C.6.1 Trouble Shooting Flowing

- **Issue: The connection between the hose barb and the needle is leaking a large amount of solvent**

- Solve: Ensure that the needle you are using only has two barbs 180^o opposite each other as opposed to four barbs 90^o apart. The long needles purchased from the stock room have four barbs , so to make them compatible with the Luer Locking system of the hose fitting you will need to file down 2 opposite barbs down. To check if this is the issue, simply replace the needles with disposable BD needles from the stock room.

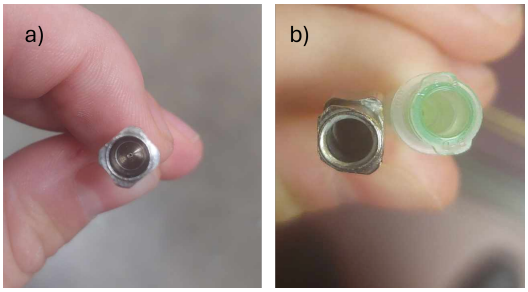

Figure C.7: A) A 4 pronged needle B) two 2 pronged luer compatible needles

- **Flow pressure decreases after several minutes**

- Solve: If you are using a septum capped vial, place a venting needle in the septum to release pressure. Too much positive pressure causes flow issues. Also, empty the flowing tubes of liquid if possible and remove the needles, sometimes particulate can get stuck in the needle inlet.

- **Tubes keep on popping off when flowing, and I sprayed solvent everywhere**

- Make sure your intake and your exit needle are the same gauge. If your exit needle is larger gauge (thinner) than your intake, solution may not be able to enter and exit at the same flow rate causing tubing to pop off

- **Flowing only persists in one direction**

- The peristaltic pump only works in one direction. Sorry!

BIBLIOGRAPHY

(1) Harrick Temperature Controled Liquid Control Cell with SwagelokTM Fittings; Harrick Scientific Products Inc, 17 pp., forthcoming.

(2) Harrick Temperature Controller Users Manual for ATK-024-3 and ATK-024-4; Harrick Scientific Products Inc, 35 pp., forthcoming.

Appendix D

THE HOME BUILT MARKOV CHAIN MONTE CARLO (MCMC) ALGORITHM

D.1 Explanation of MCMC code

Here I will go through different sections of code. If you wish to see all of these comments in the MCMC code file , you can download the file "MCMCHammer_ez.py" . In this case, I assume you already have python downloaded on your local machine, if you do not go here and download anaconda to use the SPYder Integrated Development Environment (IDE). You can use Anaconda as a package manager.

D.1.1 Imports and Function Input Variables

```
import matplotlib.animation as animation
import warnings
import tqdm as tq
import os as os
from datetime import datetime


def MCMCHammer_ezread(m_init:np.array,
                      logPfuns_in,
                      mc_count:int,
                      StepSize=2.5,
                      ThinChain:int =10,
                      ProgressBar:bool = True,
                      Parallel :bool = False,
```

```
    BurnIn=0,
    Temper:bool = False,
    T0=4000, slope=0.94,
    Save:bool = True,
    Path:str  = os.getcwd(),
    Name:str = 'MCMC_'+str(datetime.date(datetime.now())),
    Overwrite:bool=False,
    debug:bool = False,
    debug_plot:bool = True,
    debug_loop:bool = True):
```

Here we input the following python packages:

- `matplotlib.animation` as animation - allows us to animate our plots
- `warnings` - allows python to spit out a warning if our input variables could result in the algorithm crashing
- `tqdm` - prints a progress bar in the consol. Especially useful when you have runs with many iterations as well as walkers
- `os` - provides a way to interact with the operating system
- `datetime` - provides the date and time down to the seconds

The `MCMC_ezread` function takes input 17 variables that can be explained below .

- `m_init`
  - **Explanation** - The array of initial model parameters. Each column holds the parameters for each walker and each row holds a specific parameter.

  - **Data type** - np.array that is the number of parameters by the number of walkers.

- `logPfuns_in`

  - **Explanation** - The function that compute the Log probability of the model. In our implementation, we add the functions for the log priors the log likelihood to find the posterior, but you will hear more about this in Section D.2
  - **Data type** - np.array usually 3 dimensional that is 2 $\times$ shape of Likelihood $\times$ shape of prior

- `mc_count`

  - **Explanation**- Iterations of the MCMC algorithm to be completed, which determines how many samples are drawn across all walkers in parameter space
  - **Data type** - integer

- `StepSize`

  - **Explanation**- How far the MCMC moves each iteration. A larger number results in larger jumps in parameter space while a smaller number results in smaller jumps.
  - **Data type** - none-specified default 2.5

- `ThinChain`

  - **Explanation** By how much the total produced samples are culled. One out of every `ThinChain` samples is kept, with the rest discarded.

  - **Data type** - integer default 10

- `ProgressBar`
  - **Explanation**- Whether we want the progressBar to show during measurements
  - **Data type** - Boolean default True

- `Parallel`
  - **Explanation**- Whether we want parallel processing, I have not implemented this, but it would be nice
  - **Data type** - Boolean default False

- `BurnIn`
  - **Explanation**- Fraction of the samples to be discarded from the beginning. A burn in of 0.2 for example will remove the first 20% of the run
  - **Data type** - Float between 0 and 1

- `Temper`
  - **Explanation**- Feature that allows for Walkers in first iteration steps to largely be accepted. At the beginning of the sampling, walkers exist in probability space at a high "temperature" so that they can potentially get out of initial high probability local maxima. Over the course of algorithm iterations, the walkers cool off and are less accepted
  - **Data type** - Bool default false

- `T0`

  - **Explanation**- The initial temperature for tempering
  - **Data type** - float default 4000

- `Slope`
  - **Explanation**- How fast the initial hot walkers from Temper cool. A number closer to 1 results in a slower cooling.
  - **Data type** - float between 0 and 1 default 0.94

- `Save`
  - **Explanation**- If the model parameters and log probabilities get saved
  - **Data type** - Bool default true

- `Path`
  - **Explanation**- Where the fit results are saved
  - **Data type** - str default current directory as per os.getcwd()

- `Name`
  - **Explanation**- What the files will be saved as
  - **Data type** - str default 'MCMC' + the date found from datetime

- `Overwrite`
  - **Explanation**- If files of the same name will be over written
  - **Data type** - bool default False

- `debug`

- **Explanation**- Activates debug mode that tracks walker positions in parameter space and associated log probabilities
- **Data type** - bool default False

- `debug_plot`
  - **Explanation**- Whether animated plots of walkers in parameter space are plotted
  - **Data type** - bool default True

- `debug_loop`
  - **Explanation**- Whether we are tracking loop level debugging
  - **Data type** - bool default True

D.1.2 MCMC Shape Warnings and Shape setting

```
if m_init.shape[1] < 2*m_init.shape[0]:
    warnings.warn("Check m_init dimensions. It is recommended that there be
        at least twice as many walkers in the ensemble as there are model
        dimensions.")

N_walkers = m_init.shape[1]
if not N_walkers % 2 == 0:
    N_walkers -= 1
    warnings.warn("Number of walkers should be even. Removed 1 walker to
        ensure this.")

N_params = m_init.shape[0]
N_samples = np.int64(np.ceil(mc_count / N_walkers / ThinChain))
```

mc_count = (N_samples − 1) ∗ ThinChain + 1

First, we check the length of `m_init` first dimension (the number of walkers) is at least 2 times greater than the `m_init` zeroth dimension (the number of parameters). This is because for the walkers to adequately explore parameter space, there needs to be at least two times as many walkers as parameters. If the number of walkers is not at least 2 times greater than the number of parameters, a warning is raised.

Then, the number of walkers `N_walkers` is set as the initial dimension of `m_init`. We then check if this number is even, and if it is not, subtract 1 from it. Further in the algorithm, we divide the walkers into pairs in an Affine Ensemble sampler, so we need an even number of walkers. If the initial walker number was odd, a warning is thrown saying a walker was removed.

We then set the number of parameters (`N_params`) to the zeroth dimension of `m_init`. We calculate the number of samples to cycle through per walker (`N_samples`) via the following equation $N_{\text{samples}} = \left\lceil \frac{\text{mc\_count}}{N_{\text{walkers}} \times \texttt{ThinChain}} \right\rceil$. Finally, we ensure that the amount of mc iterations (`mc_count`) is divisible by the thinning parameter via the equation $\texttt{mc\_count} = (\texttt{N\_samples} - 1) \times \texttt{ThinChain} + 1$, so we do not have division issues while sampling.

D.1.3 Progress Bar Initialization

```
if ProgressBar:
    pbar = tq.trange(N_samples)
else:
    pbar = range(N_samples)
```

If the input parameter `ProgressBar` is set true, the progress bar is initialized with an interval equal to the number of samples. Otherwise, no progress bar.

D.1.4 Initializing Save Arrays

```
burn_row = np.int64(np.ceil(N_samples * BurnIn))

models = np.empty((N_params, N_walkers, np.int64(np.floor(N_samples * (1 -
    BurnIn)))))
models[:] = np.nan
models[:, :, 0] = m_init[:, :N_walkers]

logPfuns = logPfuns_in
N_funs = len(logPfuns)
logP = np.empty((N_funs, N_walkers, np.int64(np.floor(N_samples * (1 -
    BurnIn)))))
logP[:] = np.nan
```

First , we calculate the number of burnin rows by multiplying the number of samples by the `BurnIn` . Since `BurnIn` is some number between 0 and 1, we must round up to get a whole number.

Empty arrays for models and log probabilities to be filled over sampling are initialized with dimensions of `N_params` × `N_walkers` × `N_samples` excluding burn in . Then these two arrays are then filled with Nan values.

D.1.5 The Debug Block

```
if debug:
    debug_move = np.empty((N_params,N_samples*ThinChain))
    debug_zz = np.empty(N_samples*ThinChain)
    debug_partner = np.empty((N_params,N_samples*ThinChain))
    debug_propose = np.empty((N_params,N_samples*ThinChain))
```

debug_accept = np.zeros(N_samples∗ThinChain,dtype=bool)

debug_ensemble1 = np.empty((N_params,np.int64(np.ceil(N_walkers/2))−1, N_samples∗ThinChain))

debug_ensemble2 = np.empty((N_params,np.int64(np.floor(N_walkers/2)) −1,N_samples∗ThinChain))

debug_logP = np.empty(N_samples∗ThinChain)

debug_mean = np.empty((N_params,N_samples∗ThinChain))

These save arrays are only initialized if the input variable `debug` is true

- `debug_move`
  - **Explanation**- Stores the parameters (moves in parameter space) of walkers at each MCMC step
  - **Dimension** - `N_params` × (`N_samples` × `ThinChain`) The number of parameters by number of samples after being culled by the thin chain.
- `debug_zz`
  - **Explanation**- Stores the stretch factors for each iteration
  - **Dimension** - One dimension of size (`N_samples` × `ThinChain`) of a length which is the number of samples culled by chain thinning.
- `debug_partner`
  - **Explanation**- Stores parameter values of the partner walker
  - **Dimension** - `N_params` ×(`N_params` × `ThinChain`) . The number of parameters by number of samples culled by the thin chain.
- `debug_proposed`

  - **Explanation**- Stores parameter values proposed
  - **Dimension** - `N_params` $\times$ (`N_params` $\times$ `ThinChain`) . The number of parameters by number of samples culled by the thin chain.

- `debug_accept`
  - **Explanation**- Stores if proposed moves are accepted TRUE or rejected FALSE
  - **Dimension** - `N_params` $\times$ (`N_params` $\times$ `ThinChain`) . The number of parameters by number of samples culled by the thin chain. Datatype is boolean.

- `debug_ensemble1` `debug_ensemble2`
  - **Explanation**- Tracks the parameter values for each half of the walker set
  - **Dimension** - `N_parames` $\times$ `N_walkers`/2 $\times$ (`N_params` $\times$ `ThinChain`) . The number of parameters by number of samples culled by the thin chain. In this case, since the ensembles are split in half, and we subtract 1 for there not to be issues with the current iteration

- `debug_logP`
  - **Explanation**- Tracks the logP parameters for the current iteration
  - **Dimension** - One dimension of size (`N_samples` $\times$ `ThinChain`)

- `debug_mean`
  - **Explanation**- Tracks the mean value of parameters across all walkers at each step
  - **Dimension** - One dimension of size (`N_samples` $\times$ `ThinChain`) The number of samples culled by chain thinning.

D.1.6 Checking the prior probability

```
for i_walk in range(N_walkers):
    for i_func in range(len(logPfuns)):
        logP[i_func,i_walk,0] = logPfuns[i_func](m_init[:, i_walk])

if np.any(~np.isfinite(logP[:,:,0])):
    raise RuntimeError('Initial walker positions must have finite logP')
```

This section checks the probability of the priors stating that if there are any instances of - ∞ the whole algorithm raises an error. This makes sense because in the Implementation section D.2, we set priors that are true to 0 ($\log(1) = 0$ and false to - ∞. More specifically, the algorithm loops over all the walkers. Then in an internal loop, loops over the the logPfuns. Finally, we evaluate either the prior or the likelihood on the initial parameter set `M_init` for a specific walker based on the loop. This value is saved to the array LogP. If any - ∞ values are found , the MCMC is killed.

D.1.7 I'M ANGERED - Setting Up Temper

```
if Temper:
    beta = np.divide(1,T0*slope**(np.arange(N_samples))+1)
else:
    beta = np.ones((N_samples,1))
```

If Temper is true, then the slope value is raised to an array from 0 to N_samples and multiplied by the initial temper value Temperature and added to 1. This array is then reciprocated, such that beta increases over samples to the limit of 1. If temper is False, beta is just an array of 1 of the length N_samples.

D.1.8 Logging MCMC Params

```
if Save:
    model_name = Name+'_modeldata.csv'
    logP_name = Name+'_logPdata.csv'
    log_name = Name+'_log.txt'
    starttime = datetime.now()
    if Overwrite:
        with open(Path+'\\'+model_name,'w') as model_file:
            model_file.write(f'N_params: {N_params},N_walkers: {N_walkers
                }\n')
        with open(Path+'\\'+logP_name,'w') as prob_file:
            prob_file.write(f'Functions: {N_funs},N_walkers: {N_walkers}\n')
        with open(Path+'\\'+log_name,'w') as log_file:
            log_file.write(
                f"{Name} \nMCMC Log File: {str(datetime.now())}\n"
                f"Parameters: {N_params}, Walkers: {N_walkers}\n"
                f"Total MC Count: {mc_count}, ThinChain: {ThinChain},
                    Number of Samples: {N_samples}\n"
                f"Step Size Parameter: {StepSize}, Burn In Fraction: {BurnIn
                    }\n"
                )
            if Temper:
                log_file.write(f"Temper: {Temper}, Start Temp: {T0}, Slope:
                    {slope}\n")
            else:
                log_file.write(f"Temper: {Temper}\n")
```

```
elif  os.path.exists(Path+'\\'+model_name):
    file_ind  = 0
    while os.path.exists(Path+'\\'+Name+f'_modeldata({file_ind}).csv'):
        file_ind  += 1
    model_name = Name+f'_modeldata({file_ind}).csv'
    logP_name = Name+f'_logPdata({file_ind}).csv'
    log_name = Name+f'_log({file_ind}).txt'

    with open(Path+'\\'+model_name,'w') as model_file:
        model_file.write(f'N_params: {N_params},N_walkers: {N_walkers
            }\n')

    with open(Path+'\\'+logP_name,'w') as prob_file:
        prob_file.write(f'Functions: {N_funs},N_walkers: {N_walkers}\n')

    with open(Path+'\\'+log_name,'a') as log_file:
        log_file.write(
            f"{Name} \n MCMC Log File: {str(datetime.now())}\n"
            f"Parameters: {N_params}, Walkers: {N_walkers}\n"
            f"Total MC Count: {mc_count}, ThinChain: {ThinChain},
                Number of Samples: {N_samples}\n"
            f"Step Size Parameter: {StepSize}, Burn In Fraction: {BurnIn
                }\n"
            )
        if Temper:
```

```
                log_file .write(f"Temper: {Temper}, Start Temp: {T0}, Slope:
                    {slope}\n")
            else:
                log_file .write(f"Temper: {Temper}\n")

    else:
        with open(Path+'\\'+model_name,'w') as model_file:
            model_file.write(f'N_params: {N_params},N_walkers: {N_walkers
                }\n')

        with open(Path+'\\'+logP_name,'w') as prob_file:
            prob_file .write(f'Functions: {N_funs},N_walkers: {N_walkers}\n')

        with open(Path+'\\'+log_name,'a') as log_file:
            log_file .write(
                f"{Name} \n MCMC Log File: {str(datetime.now())}\n"
                f"Parameters: {N_params}, Walkers: {N_walkers}\n"
                f"Total MC Count: {mc_count}, ThinChain: {ThinChain},
                    Number of Samples: {N_samples}\n"
                f"Step Size Parameter: {StepSize}, Burn In Fraction: {BurnIn
                    }\n"
                )
            if Temper:
                log_file .write(f"Temper: {Temper}, Start Temp: {T0}, Slope:
                    {slope}\n")
            else:
                log_file .write(f"Temper: {Temper}\n")
```

This portion saves the name of the MCMC run, when it was ran, how many parameters ,how many walkers , the total iterations of the algorithm, the thinning parameter, total number of samples, the step size, burn in fractions, and whether tempering occurs.

D.1.9 Initializing Accept and Current Models and LogP arrays

```
accept = np.zeros((N_walkers, 1))
cur_m = models[:, :, 0].copy()
cur_logP = logP[:, :, 0].copy()
totcount = N_walkers
rng = np.random.default_rng()
```

This creates an accept array that is the amount of walkers long by 1 that tracks if each walker accepts a move per loop. `Cur_m` stores the current model parameters. `Cur_logP` stores the log probability of the current step. `Totcount` is equal to the number of walkers. Finally, the random number generator was initialized, for the accept reject step was initialize.

D.1.10 The crux of the MCMC

```
for row in pbar:
    for jj in range(ThinChain):


        prev_accept = np.sum(accept)
        subset_mask = np.ones(N_walkers,dtype=bool)
        subset_ind = rng.choice(np.arange(N_walkers),size = np.int64(np.floor(
            N_walkers/2)),replace = False)
        subset_mask[subset_ind] = False
        zz = ((StepSize-1)*rng.random((1,len(subset_ind)))+1)**2/StepSize
```

```
subset_cur_logP = cur_logP[:,subset_mask]

subset1_partners = rng.choice(cur_m[:,~subset_mask],len(cur_m[0,~
    subset_mask]),axis = 1)
proposed_m1 = subset1_partners - np.multiply(subset1_partners-cur_m[:,
    subset_mask],np.matmul(np.ones((N_params,1)),zz))

if debug and subset_mask[0]:
    debug_move[:,row*ThinChain+jj] = cur_m[:,np.flatnonzero(subset_mask
        )[0]]
    debug_zz[row*ThinChain+jj] = zz[0,0]
    debug_partner[:,row*ThinChain+jj] = subset1_partners[:,0]
    debug_propose[:,row*ThinChain+jj] = proposed_m1[:,0]
    debug_ensemble1[:,:,row*ThinChain+jj] = cur_m[:,np.flatnonzero(
        subset_mask)[1:]]
    debug_ensemble2[:,:,row*ThinChain+jj] = cur_m[:,np.flatnonzero(~
        subset_mask)[1:]]

if Parallel:
    print('No really, idk')
else:

    subset1_accept = np.isfinite(np.apply_along_axis(logPfuns[0],0,
        proposed_m1))

    if debug and subset_mask[0] and not subset1_accept[0]:
        debug_accept[row*ThinChain+jj] = False
```

```
            debug_logP[row*ThinChain+jj] = -np.inf

        if np.count_nonzero(subset1_accept) > 0:
            subset1_lograng = np.log(rng.random(np.count_nonzero(
                subset1_accept)))
            proposed_logP = np.array([np.apply_along_axis(logPfuns[0],0,
                proposed_m1[:,subset1_accept]), np.apply_along_axis(logPfuns
                [1],0, proposed_m1[:,subset1_accept])])

            test_accept = np.less(subset1_lograng,np.squeeze((N_params-1)*
                beta[row]*np.log(np.transpose(np.transpose(zz)[subset1_accept
                ]))+beta[row]*(np.sum(proposed_logP-subset_cur_logP[:,
                subset1_accept],0))))
            subset1_accept[subset1_accept] = test_accept
        cur_m[:,np.flatnonzero(subset_mask)[subset1_accept]] = proposed_m1[:,
            subset1_accept]
        cur_logP [:, np.flatnonzero(subset_mask)[subset1_accept]] =
            proposed_logP[:,test_accept]
        accept[np.flatnonzero(subset_mask)[subset1_accept]] += 1
    if debug and subset_mask[0] and not debug_accept[row*ThinChain+jj]:
        debug_accept[row*ThinChain+jj] = subset1_accept[0]
        debug_logP[row*ThinChain+jj] = np.sum(proposed_logP[:,0])

subset2_partners = rng.choice(cur_m[:,subset_mask],len(cur_m[0,
    subset_mask]),axis = 1)

zz = ((StepSize-1)*rng.random((1,len(subset_ind)))+1)**2/StepSize
```

```
subset_cur_logP = cur_logP[:,~subset_mask]
proposed_m2 = subset2_partners - np.multiply(subset2_partners-cur_m[:,~
    subset_mask],np.matmul(np.ones((N_params,1)),zz))

if debug and not subset_mask[0]:
    debug_move[:,row*ThinChain+jj] = cur_m[:,np.flatnonzero(~
        subset_mask)[0]]
    debug_zz[row*ThinChain+jj] = zz[0,0]
    debug_partner[:,row*ThinChain+jj] = subset2_partners[:,0]
    debug_propose[:,row*ThinChain+jj] = proposed_m2[:,0]
    debug_ensemble1[:,:,row*ThinChain+jj] = cur_m[:,np.flatnonzero(~
        subset_mask)[1:]]
    debug_ensemble2[:,:,row*ThinChain+jj] = cur_m[:,np.flatnonzero(
        subset_mask)[1:]]

if Parallel:
    print('No really, idk')
else:

    subset2_accept = np.isfinite(np.apply_along_axis(logPfuns[0],0,
        proposed_m2))

    if debug and not subset_mask[0] and not subset2_accept[0]:
        debug_accept[row*ThinChain+jj] = False
        debug_logP[row*ThinChain+jj] = -np.inf

    if np.count_nonzero(subset2_accept) > 0:
```

```
            subset2_logrand = np.log(rng.random(np.count_nonzero(
                subset2_accept)))
            proposed_logP = np.array([np.apply_along_axis(logPfuns[0],0,
                proposed_m2[:,subset2_accept]), np.apply_along_axis(logPfuns
                [1],0, proposed_m2[:,subset2_accept])])

            test_accept = np.less(subset2_logrand,np.squeeze((N_params-1)*
                beta[row]*np.log(np.transpose(np.transpose(zz)[subset2_accept
                ]))+beta[row]*(np.sum(proposed_logP-subset_cur_logP[:,
                subset2_accept],0))))
            subset2_accept[subset2_accept] = test_accept

        cur_m[:,np.flatnonzero(~subset_mask)[subset2_accept]] = proposed_m2
            [:,subset2_accept]
        cur_logP [:, np.flatnonzero(~subset_mask)[subset2_accept]] =
            proposed_logP[:,test_accept]

        accept[np.flatnonzero(~subset_mask)[subset2_accept]] += 1

    if debug and not subset_mask[0] and not debug_accept[row*ThinChain+jj]:
        debug_accept[row*ThinChain+jj] = subset2_accept[0]
        debug_logP[row*ThinChain+jj] = np.sum(proposed_logP[:,0])
    elif debug:
        debug_mean[:,row*ThinChain+jj] = np.mean(cur_m,1)
    totcount += N_walkers
```

```
if row >= burn_row:
    models [:,:, row-burn_row] = cur_m
    logP [:,:, row-burn_row] = cur_logP

    if Save and np.int64(np.floor(N_samples*(1-BurnIn))) > 10:
        if (N_samples-row) > (N_samples % 10) and (row-burn_row+1) % 10
            == 0:
            with open(Path+'\\'+model_name,'a') as model_file:
                np.savetxt( model_file ,np.reshape(np.transpose(models [:,:,( row
                    -burn_row-9):(row-burn_row+1)]),(10,N_params*
                    N_walkers)),delimiter=',')
            with open(Path+'\\'+logP_name,'a') as prob_file:
                np.savetxt( prob_file ,np.reshape(np.transpose(logP [:,:,( row-
                    burn_row-9):(row-burn_row+1)]),(10,N_funs*N_walkers)),
                    delimiter=',')

        elif  row == N_samples-1:
            with open(Path+'\\'+model_name,'a') as model_file:
                np.savetxt( model_file ,np.reshape(np.transpose(models [:,:,( row
                    -burn_row-(N_samples % 10)+1):]),((N_samples % 10),
                    N_params*N_walkers)),delimiter=',')
            with open(Path+'\\'+logP_name,'a') as prob_file:
                np.savetxt( prob_file ,np.reshape(np.transpose(logP [:,:,( row-
                    burn_row-(N_samples % 10)+1):]),((N_samples % 10),
                    N_funs*N_walkers)),delimiter=',')
```

```
        elif  Save and np.int64(np.floor(N_samples*(1-BurnIn))) <= 10 and row
            == N_samples-1:
            with open(Path+'\\'+model_name,'a') as model_file:
                np.savetxt( model_file ,np.reshape(np.transpose(models [:,:,:]) ,(np.
                    int64(np.floor(N_samples*(1-BurnIn))),N_params*N_walkers))
                    ,delimiter=',')
            with open(Path+'\\'+logP_name,'a') as prob_file:
                np.savetxt( prob_file ,np.reshape(np.transpose(logP  [:,:,:]) ,(np.
                    int64(np.floor(N_samples*(1-BurnIn))),N_funs*N_walkers)),
                    delimiter=',')
    if  ProgressBar:
        pbar.set_description(f'Acceptance: {(np.sum(accept)-prev_accept)/
            N_walkers:.1%} (Total: {np.sum(accept)/totcount:.1%})')
```

This section is rather complex and is the crux of the MCMC algorithm. It also is not super clean to read and interpret because it is written in an order that is logical to write rather than interpret. As such, I'll go through it operation by operation . By this I mean, I will talk about the bayesian inference part of this algorithm in whole before I talk about the debug part before I talk about the save part.

First, this for loop iterates over the number of samples `pbar`, and then iterates over the number of samples post thinning.

Generating and Accepting Walker Moves

`subset_mask` is a boolean array with TRUE values equal in length to the amount of walkers. `subset_ind` generates indexes from a random number generator that are replaced as false in `subset_mask`. A walker is in the first subset if its relative index is true and in the second subset if its relative index is false. Then for each proposed move , we generate a scaling factor based on the step size . A different scaling factor is

designated for each walker . We also find the current log_P for the first subset. Then , we generate `subset1_partners` partners for the first subset by randomly picking indices in the second subset.

`Proposed_m1` is the proposed positions for the first set of walkers . It is determined by finding the difference between the two sets of walkers, and then multiplying these differences by the scaling factor `zz`. Then the scaled displacement is subtracted from the partners for the final proposed position. This same process for second subset also occurs.

`Subset1_accept` is found by acting the prior probability (`logPfuns[0]`) across each collumn (parameter set) of the proposed move. If the resulting logP of the columns parameters is not finite, the column entry is labeled false. Other wise it is labeled true. If there is at least 1 accepted move from the priors, we generate the logP values for each non zero move in `Subset1_accept`. First, we generate an array of random log probabilities the same length as the number of nonzero values in `Subset1_logrand`. Then, we generate the `proposed_logP` array by applying the prior (`logPfuns[0]`) and likelihood functions (`logPfuns[1]`) on generated walkers selected via `Subset1_accept`. `proposed_logP` should be a 2 $\times$ N_Walker array dimensioned array. Then we compare `Subset1_logrand` with an acceptance threshold. To calculate the acceptance threshold we do the following calculation for each walker accepted. We find the log of the stretch parameter `zz` based on walkers accepted in `subset1_accept` and multiply this to the number of samples minus 1 . This product is then added to the sum of all difference of our proposed_logP and subset_current_logP . Finally, we scale this value by the temper value β for the sample. This acceptance threshold can be seen below in Equation D.1.

$$Threshhold = \beta_{\text{sample}} \cdot \left[(N_{\text{params}} - 1) \cdot \log \left(\prod_{i \in \text{subset1\_accept}} zz_i \right) \right. \tag{D.1}$$

$$\left. + \sum_{i \in \text{subset1\_accept}} (\text{proposed\_logP}_i - \text{subset\_cur\_logP}_i) \right]$$

If the acceptance threshold is greater than the random log probability, the walker is accepted . `Subset1_accept` is then updated with these boolean values. `cur_m` is updated with the `Proposed_m1`s that were accepted. `cur_logP` is updated with the `Proposed_logP`s that were accepted. Then, the acceptance count for each walker is updated by adding 1 if the walker was accepted. This process was repeated for the second subset of walkers as well.

Filling Debugging Arrays

If the debug boolean is set to true, and the first element of the total walker array is in the second subset mask, the following data arrays are created

- `debug_move` - This line stores the position of the first selected walker (`subset_mask`) into `debug_move`
- `debug_zz` - This line stores stretch factor of the first selected walker
- `debug_partner` - Stores the position in parameter space of the partner walker for the first walker
- `debug_propose` - Stores the proposed position of the first walker from `Proposed_m1`

`debug_ensemble` - Stores the models of the current subset that.

Then the function checks if the first walker in the total walker array is in subset 2 and that it was rejected. It then updates `debug_accept` as False and `debug_logP` as -∞.

Save Section

The algorithm checks if the sample number is higher than the amount of samples affected by the tempering , and then saves the current models and current logP to

`models` and `logP` respectively. Then , if there are more than 10 samples, the data is reshaped as a 2D array and every 10 iterations it saves a chunk of 10 samples. If there are less than 10 samples, all samples are simply saved. Then the progress bar is updated .

Debug Plotting

```
if debug and debug_plot:
    debug_yes = np.zeros((2, np.count_nonzero(debug_accept) + 1))
    debug_yes[:, 0] = [0, np.sum(logP[:, 0, 0])]
    debug_no = np.zeros((2, np.count_nonzero(~debug_accept)))

for ii in range(len(debug_accept)):
    if debug_accept[ii]:
        debug_yes[:, np.count_nonzero(debug_accept[:(ii + 1)])] = [np.
            count_nonzero(debug_accept[:(ii + 1)]), debug_logP[ii]]
    elif not debug_accept[ii] and np.isfinite(debug_logP[ii]):
        debug_no[:, np.count_nonzero(~debug_accept[:(ii + 1)]) - 1] = [np.
            count_nonzero(debug_accept[:(ii + 1)]), debug_logP[ii]]
    else:
        debug_no[:, np.count_nonzero(~debug_accept[:(ii + 1)]) - 1] = [np.
            count_nonzero(debug_accept[:(ii + 1)]), -10]

debug_fig, (debug_ax1, debug_ax2) = plt.subplots(1, 2)
artists = [
    debug_ax1.plot([], [], marker='o', color='silver', alpha=0.5)[0],
    debug_ax1.plot([], [], marker='o', color='crimson')[0],
    debug_ax1.plot([], [], marker='o', color='deepskyblue')[0],
```

```
    debug_ax1.plot([], [], linestyle='--', color='gray')[0],
    debug_ax1.plot([], [], marker='D', color='indigo')[0],
    debug_ax1.plot([], [], marker='o', color='palevioletred', ls='none',
        alpha=0.05)[0],
    debug_ax1.plot([], [], marker='o', color='palevioletred', ls='none')[0],
    debug_ax1.plot([], [], marker='o', color='steelblue', ls='none', alpha
        =0.05)[0],
    debug_ax1.plot([], [], marker='o', color='steelblue', ls='none')[0],
    debug_ax1.plot([], [], marker='P', color='black', ls='none', ms=8)[0],
    debug_ax2.plot([], [], marker='x', color='black', ls='none')[0],
    debug_ax2.plot([], [], marker='o', color='crimson')[0]
]

def update_plot(i_step, artists, debug_move, debug_zz, debug_partner,
    debug_propose, debug_accept, debug_e1, debug_e2, debug_logP, debug_yes,
    debug_no):
    artists[0].set_data(debug_move[:2, :(i_step + 1)])
    artists[1].set_data(debug_move[:2, i_step])
    artists[2].set_data(debug_partner[:2, i_step])

    if debug_zz[i_step] >= 1:
        artists[3].set_data(np.stack([debug_propose[:2, i_step].T,
            debug_partner[:2, i_step].T], axis=1))
    else:
        artists[3].set_data(np.stack([debug_move[:2, i_step].T, debug_partner
            [:2, i_step].T], axis=1))
```

```
artists [4].set_data(debug_propose[:2, i_step])

if not debug_accept[i_step]:
    artists [4].set_marker('x')
else:
    artists [4].set_marker('D')

if i_step > 0:
    artists [5].set_data(np.stack([debug_e1[:2, :, ii] for ii in range(
        i_step)], axis=1))
else:
    artists [5].set_data ([], [])

artists [6].set_data(debug_e1[:2, :, i_step])

if i_step > 0:
    artists [7].set_data(np.stack([debug_e2[:2, :, ii] for ii in range(
        i_step)], axis=1))
else:
    artists [7].set_data ([], [])

artists [8].set_data(debug_e2[:2, :, i_step])
artists [9].set_data(debug_mean[:2, i_step])

if i_step == 0:
    artists [10].set_data ([], [])
    artists [-1].set_data(debug_yes[:, 0])
```

```
        if debug_accept[i_step]:
            artists[-1].set_data(debug_yes[:2, :(np.count_nonzero(debug_accept[:(
                i_step + 1)]) + 1)])
        else:
            artists[10].set_data(debug_no[:2, :np.count_nonzero(~debug_accept[:(
                i_step + 1)])])

        return artists

    num_frames = N_samples * ThinChain
    global debug_ani
    debug_ax1.set(title='Ensemble', xlim=(np.min(np.hstack((debug_move[0, 0],
        np.ravel(debug_ensemble1[0, :, :])))) - 1, np.max(np.hstack((debug_move[0,
         0], np.ravel(debug_ensemble1[0, :, :])))) + 1), ylim=(np.min(np.hstack((
        debug_move[1, 0], np.ravel(debug_ensemble1[1, :, :])))) - 1, np.max(np.
        hstack((debug_move[1, 0], np.ravel(debug_ensemble1[1, :, :])))) + 1))
    debug_ax2.set(title='Walker 1 Log(P)', xlim=(-1, np.count_nonzero(
        debug_accept) + 1), ylim=(np.min(debug_logP[np.isfinite(debug_logP)]) -
        0.5, np.max(debug_logP[np.isfinite(debug_logP)]) + 0.5))
    debug_ani = animation.FuncAnimation(debug_fig, update_plot, num_frames,
        fargs=(artists, debug_move, debug_zz, debug_partner, debug_propose,
        debug_accept, debug_ensemble1, debug_ensemble2, debug_logP, debug_yes,
        debug_no), interval=500, repeat=debug_loop)
    plt.show()
```

This block of code produces two plots: a 2D ensemble plot of all the walkers in parameter space and a logP over time for one walker. The `debug_yes` array is

filled with accepted moves and the `debug_no` array is filled with rejected moves. The `Artists` array generates markers of different colors for each walker as well as 2 subplots one for the walker space and one for the logP. Then the update plot function is run and updates for each sample, only showing thinned by thin chain samples.

Saving and Returning Models, logP

```
    if Save:
        with open(Path + '\\' + log_name, 'a') as log_file :
            log_file .write(f'Complete. Run Time: {datetime.now() - starttime}\
                n' '
                ----------------------------------------------------------------
                n')

    if debug:
        return models, logP, debug_move, debug_propose, debug_accept

    return models, logP
```

If the results of the MCMC are to be save, a logfile is saved. If debug is activated, `models`,`logP`, `debug_move` - the current position of walkers, `debug_proposed` - the proposed position of walkers, and `debug_accept` - whether the walkers are accepted or rejected are returned. Other wise `models`,`logP` are just returned.

D.2 Implementation of the MCMC Algorithm

You have just read the explanation of the spaghetti code that is the MCMC algorithm, and probably are thinking in your head "How the heck do I actually use the algorithm ???". Fear not young Padawan, in this section I will explain the ins and outs of implementation. In general, to get good understanding of the posterior distribution

one should run the MCMC with as many walkers and as many iterations as possible, so that the walkers can find the local probability maxima. However, in real life, we are limited by our computers, so I recommend sampling with fewer iterations and walkers initially to find out where in parameter space the parameters lie, then adjust the prior and likelihood functions as well as the amount of walkers and iterations.

Here I will go through fitting an arbitrary 2 gaussian function denoted by the following equation D.2 with parameters listed in Table D.1 where amplitude is amplitude, σ is width, and μ is center. The resulting plot is seen in D.1.

$$y = \text{amp}_1 \cdot \exp\left(-\frac{(f-\mu_1)^2}{2\sigma_1^2}\right) + \text{amp}_2 \cdot \exp\left(-\frac{(f-\mu_2)^2}{2\sigma_2^2}\right) \tag{D.2}$$

	Amplitude	σ	μ
Gauss 1	0.51	19.5	1584
Gauss 2	1.27	15.718	1517.5

Table D.1: Parameters for Gaussian fits

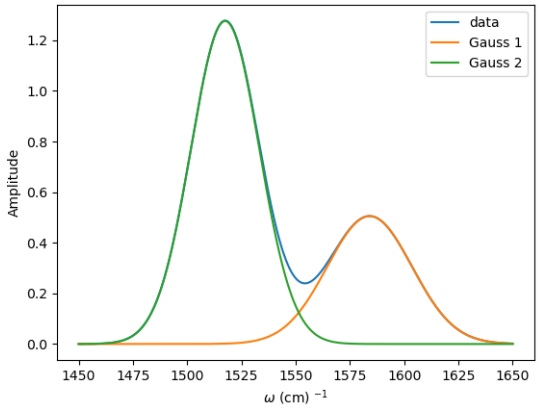

Figure D.1: The constructed 2 gaussian function

D.2.1 Loading in python imports and MCMC Parameters

```
import sys
sys.path.insert(1, home_dir + '/OneDrive - UW/UW-all years/
    MCMCHammer_V3/')
import numpy as np
from GenInitBall_2 import GenInitBall
from MCMCHammer_v0p2 import MCMCHammer
import matplotlib.pyplot as plt
from MCMC_Plot import MCMC_Plot
from MCMC_plot_better import MCMC_Plot_better



temper = False
#BurnIn
burn = 0
```

count = 100000

walk = 30

step = 1.6

rng = np.random.default_rng()

Here we load in several functions such as MCMCHammer , which we discussed above in D.1 and Matplotlib for plotting. We also import in a function `GenInitBall` to generate initial parameters in an N dimensional ball. This is further explained in Section D.2.2. We load in our variables for whether we burn in walkers (`burn`), number of iterations (`count`) ,number of walkers (`walk`), and stepsize (`step`). Here we initialize a MCMC run with no burn in 30 walkers, 100,000 time points, and a step size of 1.6. This step size of 1.6 was used to keep the acceptance parameter between 20% and 40 % as per the MCMC Diagnostics section ()

D.2.2 Generating Parameters

To generate parameters initially for the MCMC we use the following function . The function `GenInitBall` is designed to generate a set of random points (walkers) within an N-dimensional ball. The function takes as input the coordinates for the center of the ball in each dimension, the radii for each dimension, and the number of random points (walkers) to generate.

GenInitBall

---

import numpy as np

def GenInitBall(centers, radii , N_walkers):

```
    if len(centers) != len(radii):
        raise RuntimeError(f'Inconsistent dimensions between center ({len(centers)
            }-dim)'
                           f' and radii ({len(radii)}-dim)')
    centers = np.array(centers)
    radii = np.array(radii)
    N_dim = len(centers)
    rng = np.random.default_rng()

    u_mat = rng.normal(size=(N_walkers, N_dim))
    norm = np.linalg.norm(u_mat, axis=1, keepdims=True)
    u_mat /= norm

    rad = rng.random(size=(N_walkers, 1))**(1.0 / N_dim)
    ball_mat = u_mat * rad

    x_mat = centers + ball_mat * radii

    return x_mat.T

amp_init = GenInitBall ([3,3],[3,3], walk)

sigs_init = GenInitBall ([12,12],[8,8], walk)

means_init = GenInitBall ([1550,1550],[50,50], walk)
```

m_init = np.vstack((amp_init [0,:], sigs_init [0,:], means_init [0,:], amp_init [1,:], sigs_init [1,:], means_init [1,:]))

Inputs

- `centers`: A list or array of the coordinates for the center of the ball for each dimension.This must be equal in length to the input radii array, and determines the ball dimensionality N_{dim}.

- `radii`: A list or array of the coordinates for the radii of the ball for each dimension. This must be equal in length to the input `centers` array

- `N_walkers`: The number random points in parameter space , also known as walkers to generate in the N-dimensional ball.

Output

- The function returns an $N_{\text{dim}} \times N_{\text{walkers}}$ matrix (`numpy.ndarray`) where each column represents a point's coordinates within the N-dimensional ball.

`GenInitBall` first converts the input `centers` and `radii` to `numpy` arrays and checks that their lengths match. It then determines the dimensionality of the space, N_{dim}, based on the length of `centers`. To generate random points, the function creates a matrix `u_mat` of size $(N_{\text{walkers}}, N_{\text{dim}})$ with elements drawn from a standard normal distribution (mean 0, variance 1). Each row of `u_mat` is normalized to lie on the surface of an N-dimensional unit sphere by dividing by its Frobenius norm. To ensure that the points are uniformly distributed within the sphere, the function generates an array `rad` of random values between 0 and 1, which are then raised to the power $\frac{1}{N_{\text{dim}}}$. The resulting matrix, `ball_mat`, is found by scaling `u_mat` with `rad`, producing points distributed throughout the unit sphere. These points are then adjusted to match the

desired dimensions by scaling them with the input `radii` and shifting them according to the `centers`. Finally, the function returns the transposed matrix, where each column represents the coordinates of a point within the N-dimensional ball.

Using `GenInitBall` we generate 3 parameter arrays, one for amplitude, σ, and μ that are $N_{gaussians}$ by $N_{walkers}$. Then we stack the generated parameters on top of each other, with the parameters for the first gaussian first and the parameters for the second gaussian second. NOTE: one should be consistent about what order they put their parameters in, or else you will get proposed widths where your proposed centers should be or something equally devastating.

D.2.3 Linear Least Squared

This algorithm employs linear least squared matrix algebra. As a reminder, if we were to want to fit some n observable by m parameter matrix **A** to some vector y, we would write the equation

$$A\beta = y$$

where β is a matrix of fit parameters, and y is the vector we want to fit to. If Equation C.6 is consistent, the best fit equation is just the solution of Equation D.2.3. Other wise, we seek to minimize the quantity $||Ax - y||^2$. To do this, we normalize Equation D.2.3 via

$$A^T A\beta = A^T y \tag{D.3}$$

where A^T is the transpose of A. Solving for the β fit parameters, we multiply the $(A^T A)^{-1}$ by both sides to yield.

$$\beta = (A^T A)^{-1} A^T y \tag{D.4}$$

D.2.4 Generating the Likelihood function

```
def gauss1(f, amp, sig, w0):
    y = amp * np.exp(-(f-w0)**2/2/sig**2)

def gauss_2(f, y_data,amp1, sig1, w1,amp2, sig2, w2):
    lsq_mat = np.hstack([np.reshape(gauss1(x,amp1,sig1,w1),(-1,1)),
                         np.reshape(gauss1(x,amp2,sig2,w2),(-1,1)),])

    try:
        beta_mat = np.abs(
            np.matmul(
                # the below line is (model_function^T*model_function)^-1 *
                    model_function^T
                np.matmul(np.linalg.inv(np.matmul(np.transpose(lsq_mat),lsq_mat))
                    ,np.transpose(lsq_mat)),
                #the below line multiplies the result of the above line by the
                    data matrix
                np.reshape(y_data,(-1,1))))
        # cumulatively beta_mat = (model_function^T*model_function)^-1 *
            model_function^T*data
    except np.linalg.LinAlgError as err:
        if 'Singular matrix' in str(err):
            #Error handling section
            beta_mat = np.abs(np.linalg.lstsq(lsq_mat,np.reshape(y_data,(-1,1)),
                rcond=None)[0])
        else:
            raise
    # generated data from model function and data matrix in transmission
```

```
    y_data = np.matmul(lsq_mat,beta_mat)
    return y_data
def likelihood(params):
        residual = gauss_2(x,y,*params).flatten()-y
        return -1*np.sum(residual**2)
```

Our likelihood function will be the negative of the sum of residuals squared between the actual data and the model function . In this case, we define our model function as the sum of two gaussians, so we define a function `gauss1` as a gaussian function of amplitude , σ as width, and μ as center. Then `gauss1` is used to make x row $\times$ 2 collumn A matrix, where x is the length of the data to be fit and 2 is the amount of gaussians to fit to the data, denoted as Y henceforth . In the code above, the A matrix is denoted as `lsq_mat`. Note: usually this A matrix has the same number of collumns as your fit function has terms, so 3 gaussians would be 3 collumns and a linear fit (with slope and intercept terms) would also be 2. If you have a fit that requires some offset i $x^2 + c$, I recommend writing a column full of ones for the offset.

In the above code, we use a try and except statement, to do the matrix multiplication of

$$\beta = \left(\texttt{lsq\_mat}^{\mathrm{T}}\texttt{lsq\_mat}\right)^{-1}\texttt{lsq\_mat}^{\mathrm{T}}y$$

to find the fitting parameters from the y and `lsq_mat`. If `lsq_mat` cannot be inverted, we use the python package to solve the linear equation

$$[\texttt{lsq\_mat}]\beta = y$$

,

Multiplying our `lsq_mat` by β yields our fit. From here, we simply subtract the fit from our actual data y, to find the residuals. The negative of the square of the residuals will be equal to the likelihood.

D.2.5 Generating the Prior Function

Uninformed Priors

```
def prior(params):
    if np.all(np.isfinite(params)):
        logprob = 0;
    else:
        logprob = -np.Inf
    return logprob
```

The uninformed prior is relatively simple. For uninformed priors, this function should evaluate True to 0 and false to -Inf. This is because we're using the log prior, so a probability of 1 maps to log(1) = 0, and a probability of 0 maps to log(0) -¿ -Inf

Informed Priors

Sometimes informed priors are necessary , when you know your data must have certain characteristics. For example, if you are fitting an IR spectra where you know your analyte must vibrate between 1500 and 1520 cm^{-1} you could set your MCMC to only sample in that region via informed priors

```
def prior(params):
    params = np.array(params)

    # Define lower and upper bounds for 2 Gaussians
    lower_check = (
        np.all(params[[0, 3]] > np.array([0, 0])) and # Amplitudes
        np.all(params[[1, 4]] > np.array([7, 7]))and # widths
        np.all(params[[2, 5]] > np.array([1495, 1550]))# centers
    )
```

```
upper_check = (
    np.all(params[[0, 3]] < np.array([25, 25])) and # Amplitudes
    np.all(params[[1, 4]] < np.array([50, 50])) and # widths
    np.all(params[[2, 5]] < np.array([1530, 1620]))    # centers
)

# Ensure that the centers are in increasing order
order_check = params[2] < params[5]

# Additional condition check
if lower_check and upper_check and order_check:
    return 0
else:
    return -np.inf
```

In this case, we want to ensure that our amplitudes are greater than 0 and less than 3 , our widths are between 7 and 50, and our centers are between 1495 and 1530 and 1550 and 1620 respectively. NOTE: for informed priors you must make sure that your generated initial parameters from `GenInitBall` are within the range established by your priors or else the MCMC will kill itself. This prior function also ensure that the two generated peaks are in order and not swapping out.

D.2.6 Initializing the MCMC Algorithm

Here we initialize the MCMC algorithm with the debug feature , a chain thinning of 10. The results will not be saved. The debut plot can be seen in Figure D.2

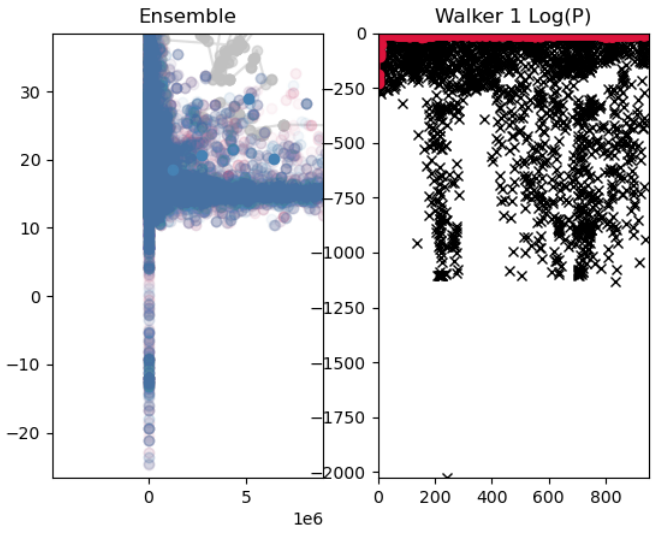

Figure D.2: The debug plot produced from MCMCHammer. On the left, we see the covariance of Amp1 with $\sigma 1$. On the right, we see how the first walkers logP changes over iteration. Xs are unaccepted parameters , and red dots are accepted

---

models3, logP3, debug_move3, debug_propose3, debug_accept3 =
MCMCHammer(m_init,(prior,likelihood),count,StepSize=step,ThinChain
=10,BurnIn = burn,Temper=temper,Save=False,debug=True)

D.2.7 MCMC Diagnostics

These diagnostics are necessary to determine how suitable are generated parameters are at describing a distribution.

Acceptance Portion

A first method of diagnostics is simply the acceptance probability, which should be between 0.2 and 0.5.[1,2] A lower acceptance probability than 0.2 will not accept enough samples for the sampling will not be representative of the target density. A higher

acceptance probability than 0.5 will have no regard for the actual target density and simply be wandering in parameter space. For each iteration of the algorithm , the acceptance probability will be displayed, and it is found via Equation

$$Acceptance = \frac{\sum Acceptance_{current} - \sum Acceptance_{previous}}{N_{walkers}} \tag{D.5}$$

where $Acceptance_{current}$ and $Acceptance_{current}$ are arrays denoting the number of times each walker was accepted.

Corner Plots

First, for each plot on the diagonal shows the marginalized distribution of a parameter. Each non diagonal plot shows how parameters vary with each other. Ideally, each covariance plot is circular suggesting that the two parameters in comparison are not correlated. Similarly, elongated covariance plots suggest correlation. Below is the corner plot for the target distribution of the 2 gaussian dataset.

Looking at this corner plot distribution, we see that there are two completely separate distributions for $\mu 1$ and $\mu 2$ one around 1520 and one around 1580. From table D.1 we know our our two μ values are $\mu 1$=1517 and $\mu 2$ = 1584. We did not set any specification when we generated parameters, so `GenInitBall` generated the two gaussians parameters negligent of the order, ergo we see highly correlated corner plots.

When we use the informed priors from the informed priors section , we see far different distributions. In many cases, we see two dimensional gaussians in the covariance plots between many parameters. While, we do see some tilted covariances implying some correlation between different parameters, we by and large are satisfying the expected normal distribution of probabe parameters.

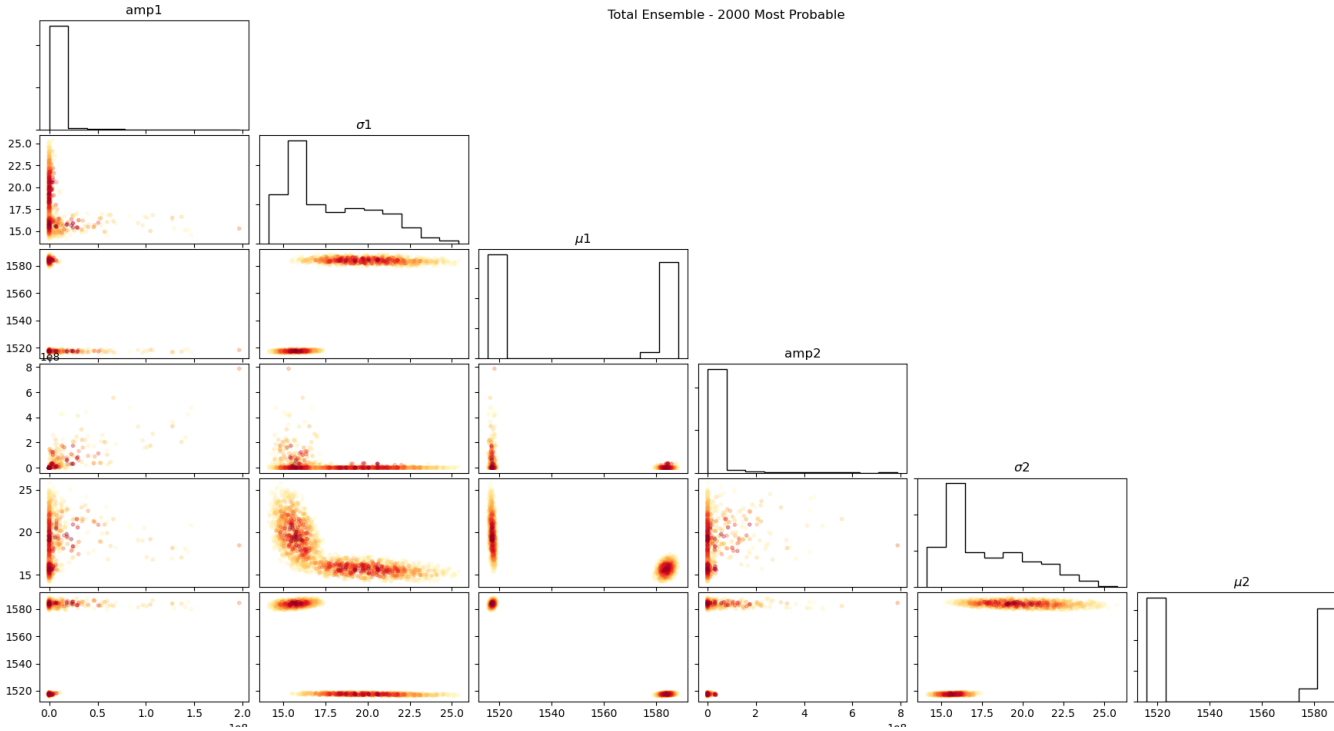

Figure D.3: Corner plot for parameters amp1, $\sigma 1$, $\mu 1$,amp2, $\sigma 2$, $\mu 2$ generated from an MCMC sampling with uninformed priors. Here we see bimodal distributions in the covariance plots indicated two populations of parameters with correlated distributions

.

The following code was used to generate Figure D.3 and Figure D.4.

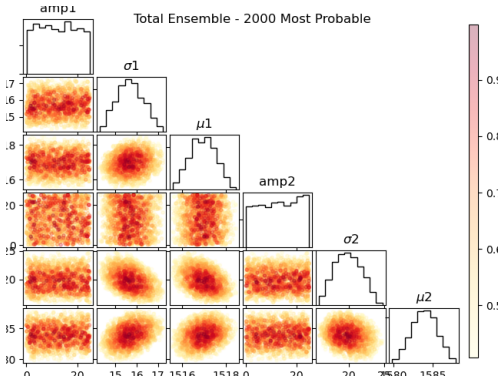

Figure D.4: Corner plot for parameters amp1, $\sigma 1$, $\mu 1$,amp2, $\sigma 2$, $\mu 2$ generated from an MCMC sampling with informed priors. Here we see two dimensional gaussian distributions around probable parameters , which can be used to calculate deviation .

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

def MCMC_Plot_better(models, logP, labels=None, overplot=False, like=False):
    N_params = models.shape[0]

    # Determine the order of the parameters based on likelihood
    if like :
        total_order = np.argsort(np.ravel(logP[1, :, :]) )
        final_order = np.argsort(logP[1, :, -1])
    else:
        total_order = np.argsort(np.ravel(np.sum(logP, axis=0)))
        final_order = np.argsort(np.sum(logP[:, :, -1], axis=0))
```

```
# Limit to 2000 most probable parameter sets if there are too many
if models.shape[1] * models.shape[2] > 2000:
    warnings.warn('Too many points! Only taking the 2000 most probable to
        avoid making the computer cry')
    total_order = total_order[-2000:]

# Set up total and final corner plots
total_fig , total_ax = plt.subplots(N_params, N_params)
final_fig , final_ax = plt.subplots(N_params, N_params)

if overplot:
    total_ax0 = total_fig.add_subplot(1, 1, 1)
    final_ax0 = final_fig.add_subplot(1, 1, 1)
    for ax in [total_ax0, final_ax0]:
        ax.set_axis_off()
        ax.set_xlim(0, N_params)
        ax.set_ylim(0, N_params)

total_fig.subplots_adjust(left=0.035, right=0.92, top=0.965, bottom=0.035,
    wspace=0.05, hspace=0.05)
final_fig.subplots_adjust(left=0.035, right=0.92, top=0.965, bottom=0.035,
    wspace=0.05, hspace=0.05)

# Calculate weights for histogram and scatter plots
if like:
    weight_rav = np.exp(np.ravel(logP[1, :, :])[total_order])
```

```
        weight_reg = np.exp(logP[1, :, -1])
        weight_fin = np.exp(logP[1, :, -1][final_order])
    else:
        weight_rav = np.exp(np.ravel(np.sum(logP, axis=0))[total_order])
        weight_reg = np.exp(np.sum(logP[:, :, -1], axis=0))
        weight_fin = np.exp(np.sum(logP[:, :, -1], axis=0)[final_order])

    # Generate plots
    for row in range(N_params):
        for column in range(row + 1):
            if row == column:
                bins = total_ax[row, column].hist(np.ravel(models[row, :, :])[
                    total_order],
                                                  density=True, weights=
                                                      weight_rav, histtype='step',
                                                      color='k')[1]
                if overplot:
                    param_mean = np.average(np.ravel(models[row, :, :])[
                        total_order], weights=weight_rav)
                    x_val = (param_mean - bins[0]) / (bins[-1] - bins[0])
                    total_ax0.plot([row + x_val] * 2, [0, N_params - row], ls='
                        --', color='tab:gray')
                    total_ax0.plot([0, N_params - row], [row + x_val] * 2, ls='
                        --', color='tab:gray')

                bins = final_ax[row, column].hist(models[row, :, -1],
                                                  density=True, weights=
```

```
                                                    weight_reg, histtype='step',
                                                    color='k')[1]
        if overplot:
            param_mean = np.average(models[row, :, -1], weights=
                weight_reg)
            x_val = (param_mean - bins[0]) / (bins[-1] - bins[0])
            final_ax0.plot([row + x_val] * 2, [0, N_params - row], ls='
                --', color='tab:gray')
            final_ax0.plot([0, N_params - row], [row + x_val] * 2, ls='
                --', color='tab:gray')

        if row != N_params - 1:
            final_ax[row, column].set_xticklabels([])
            total_ax[row, column].set_xticklabels([])
        final_ax[row, column].set_yticklabels([])
        total_ax[row, column].set_yticklabels([])

        if labels:
            total_ax[row, column].set_title(labels[row])
            final_ax[row, column].set_title(labels[row])

    else:
        total_im = total_ax[row, column].scatter(np.ravel(models[column,
            :, :])[total_order],
                                                 np.ravel(models[row, :,
                                                     :])[total_order],
                                                 c=weight_rav, cmap='
```

```
                                                    YlOrRd', alpha=0.3,
                                                    s=9)
            final_im = final_ax[row, column].scatter(models[column,
                final_order, -1],
                                                    models[row, final_order,
                                                        -1],
                                                    c=weight_fin, cmap='
                                                        YlOrRd', alpha=0.3,
                                                        s=9)

            if row < N_params - 1:
                final_ax[row, column].set_xticklabels([])
                total_ax[row, column].set_xticklabels([])
            if column > 0:
                final_ax[row, column].set_yticklabels([])
                total_ax[row, column].set_yticklabels([])

for column in range(N_params):
    for row in range(column):
        final_ax[row, column].axis('off')
        total_ax[row, column].axis('off')

# Add colorbars and titles
total_cbar = total_fig.add_axes([0.94, 0.05, 0.02, 0.9])
total_fig.colorbar(total_im, cax=total_cbar)
total_fig.suptitle('Total Ensemble - 2000 Most Probable' if models.shape[1]
    * models.shape[2] > 2000 else 'Total Ensemble')
```

```
final_cbar = final_fig.add_axes([0.94, 0.05, 0.02, 0.9])
final_fig.colorbar(final_im, cax=final_cbar)
final_fig.suptitle('Final Ensemble')

return
```

LogP values of the posterior distribution are ordered from best to worst for the entire ensemble and the final ensemble. Only the top 2000 most probable points of the ensemble are kept. The weights are calculated for plotting based on exponentiating LogP values thereby getting them back into 0 to 1 probability space. Weights are calculated for the entire ensemble and the final ensemble. Diagonal plots are histograms showing the distribution of the ensemble parameters based on the weights . Off diagonal plots show the correlation between parameters with walkers colored by their weights. Weights closer to 1 are redder and weights closer to 0 are yellower.

Autocorrelation

In an MCMC algorithm, each sample depends on samples prior to it. Consecutive samples are more likely to be similar to each other, which is called autocorrelation. If samples are high in autocorrelation, they carry similar information and the chain could easily get caught in a relative minima that does not truly depict the target ensemble. As such , comparing samples over the course of several iterations allows for determination if the posterior fits the target distribution. The quicker the autocorrelation dies from 1 to close to 0, the fewer amount of samples we need to generate a posterior that fits the target distribution.

Figure D.7 is the plot of the autocorrelation for each of the parameters, which shows that the autocorrelation decreases from 1 to near zero with a lag of approximately 40 iterations. Lag is simply an interval of separation between samples.

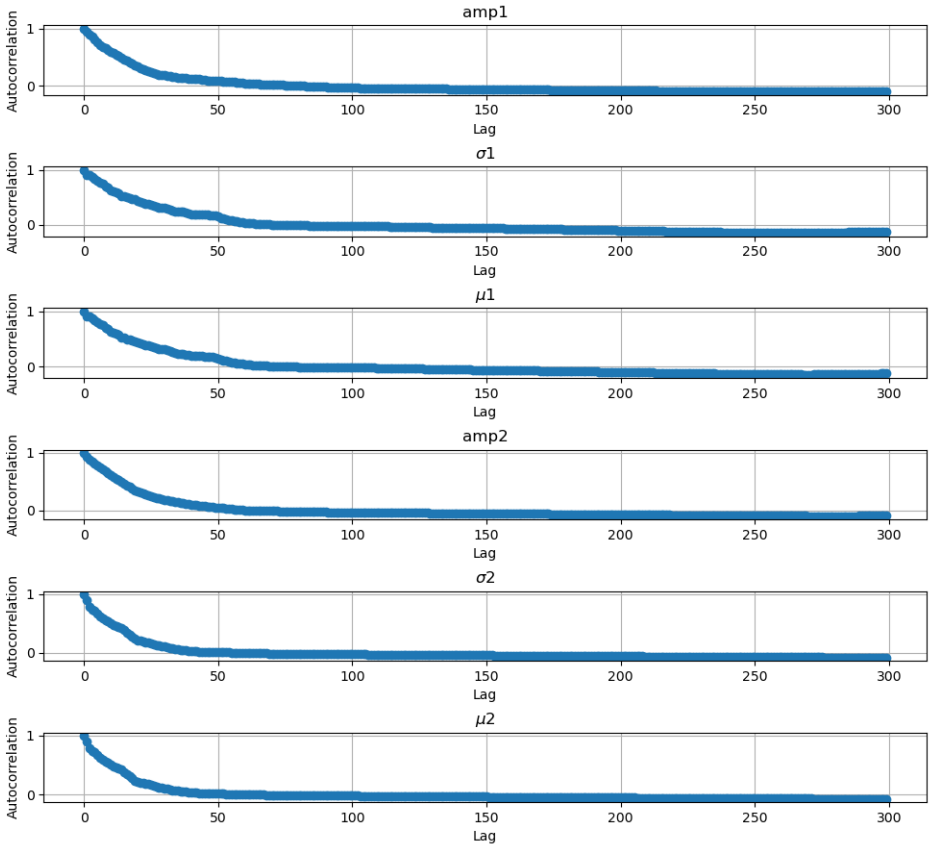

Figure D.5: Autocorrelation plot for parameters amp1, $\sigma 1$, $\mu 1$,amp2, $\sigma 2$, $\mu 2$

```
def autocorrelation(x, lag):
    """
    Calculate the autocorrelation for a given lag.
    """
    n = len(x)
    x_mean = np.mean(x)
    autocorr = np.correlate(x - x_mean, x - x_mean, mode='full')[n-1:] / np.var(
        x) / n
```

```
    return autocorr[:lag]

def plot_autocorrelation(models,paramnames, max_lag=34,):
    """
    Plot the autocorrelation function for each parameter in the MCMC output.

    Parameters:
        models (np.ndarray): MCMC output array with shape (parameters, walkers
            , samples).
        max_lag (int): Maximum lag to calculate autocorrelation.
    """
    n_params = models.shape[0]
    fig, axs = plt.subplots(n_params, 1, figsize=(10, 3 * n_params))
    fig.suptitle("Autocorrelation Functions for Each Parameter", y=1.02)

    for i in range(n_params):
        # Average over all walkers for the given parameter to get a single chain
        chain = models[i].mean(axis=0)

        # Calculate autocorrelation up to the specified max lag
        acf = autocorrelation(chain, max_lag)

        # Plot the autocorrelation function
        axs[i].plot(range(max_lag), acf, marker='o', linestyle='-')
        axs[i].set_title(paramnames[i])
        axs[i].set_xlabel('Lag')
        axs[i].set_ylabel('Autocorrelation')
```

```
        axs[i].grid(True)

    plt.tight_layout(rect=[0, 0, 1, 0.98])
    plt.show()

    return acf
```

This plots the autocorrelation for each parameter. `np.correlate` is used to calculate correlation based on lag. Note the `max_lag` must be one less than the total number of samples eg if you have 30 samples total, your `max_lag` should 29.

D.2.8 Other Useful MCMC plots and scripts

The following line orders the average logP values in ascending order

```
    total_order_sum = np.argsort(np.ravel(np.sum(logP,axis=0)))
```

The following orders the best 100 parameters, inputs them into the model function, and then averages the resulting traces

```
    total_order_params = np.reshape(models3,([N_params,-1][:,total_order_sum
        [-100:]]
best_traces = []
for ii,paramset in enumerate(total_order_params.T):
    best_traces.append(gauss_2(x,y,*paramset))
```

The following averages the best 100 parameters from the sampling with uninformed priors and inputs them into the model function. Then the data from the average parameters is plotted against the true data, data generated from the best parameter, and data generated from the 100 best parameters. Note: since the MCMC does not care about keeping gausses in order the average parameter data does not fit the target distribution at all.

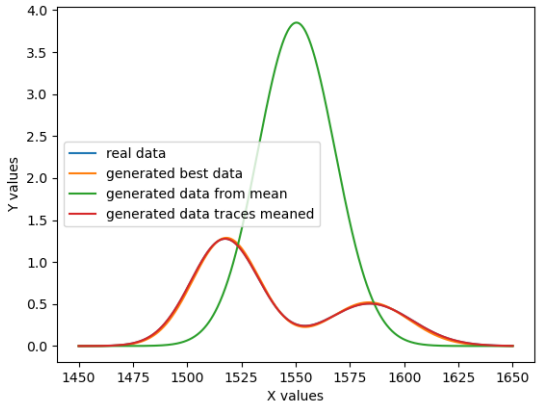

Figure D.6: Generated traces based on parameters from the MCMC . We see real data in blue, a trace generated from the walker with the lowest logP in orange, a trace generated from the average of the 100 best walkers in green, and average best 100 traces from walkers in red.

```
total_mean_sum = np.average(np.reshape(models3,([N_params,-1]))[:,
    total_order_sum[-100:]],axis=1)
best_traces  = np.array(best_traces).squeeze()
mean_traces = np.mean(best_traces,axis = 0)
best_term = gauss_2(x,y,*total_order_params [:,0])
mean_terms = gauss_2(x,y,*total_mean_sum)

plt.figure()
plt.plot(x,y, label = 'real data')
plt.plot(x,best_term,label='generated best data')
plt.plot(x,mean_terms , label = 'generated data from mean')
plt.plot(x,mean_traces, label ='generated data traces meaned')
plt.legend()
```

This plots the LogP of all walkers over all samples .

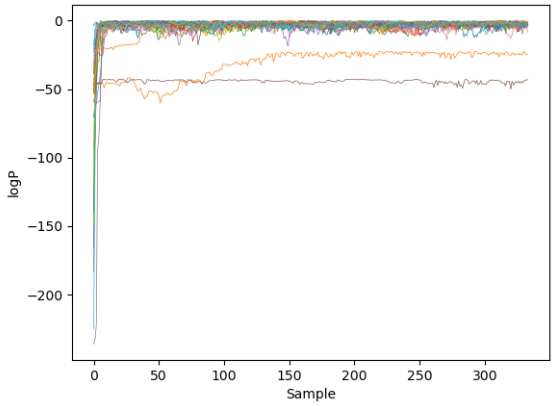

Figure D.7: logP values over samples for all walkers

```
plt.figure()
plt.plot(np.arange(len(models3[0,0,:]) ),np.transpose(np.sum(logP3,axis=0)),lw
    =0.5)
plt.xlabel('Sample')
plt.ylabel('logP')
```