Developing an Acquisition and Analytical Ecosystem for Protein Bioanalysis Using Mass Spectrometry and Programmed Temperature Control

Elizabeth Anne Fawcett

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

Master of Science

University of Washington
2025

Committee:
Matthew F. Bush, Chair
Nicholas M. Riley

Program Authorized to Offer Degree:
Applied Chemical Sciences and Technologies

University of Washington

Abstract

Developing an Acquisition and Analytical Ecosystem for Protein Bioanalysis Using Mass Spectrometry and Programmed Temperature Control

Elizabeth Anne Fawcett

Chair of the Supervisory Committee:
Matthew F. Bush
Department of Chemistry

Temperature-controlled electrospray ionization (tcESI) coupled with mass spectrometry (MS), which allows for the measurement of mass-to-charge (*m/z*) ratios, enables the analysis of both structural information and stability in proteins, specifically by evaluating their thermal denaturation, a measure of protein unfolding. In this work, programmed-temperature electrospray ionization (ptESI) source, is used to continuously and rapidly heat and cool a nanoESI capillary containing proteins in solution. Combining MS and ptESI, enables the tracking of structural changes in unfolding and refolding events from solution-phase chemistry. In ion mobility (IM) measurements, charged ions under the influence of an electric field experience collisions that allow for separation based on size, shape, and charge, which yields additional insights into protein structure. During collision-induced unfolding (CIU), collisional activation allows IM-MS to probe gas-phase ion structures regarding their stability. Shifts in stability, when assessing collisional activation, may be reflective of structural changes.

In chapter 1 of this work, the tools traditionally used for assessing protein structure and stability are described. In chapter 2, the acquisition software updates, supporting usability and maintenance, for the ptESI source are characterized. Next, in chapter 3, the programmatic analysis of the structural and stability information generated using the acquisition software is described. Then, in chapter 4, experiments utilizing ptESI, CIU, and ptESI coupled with CIU are characterized. Finally, in chapter 5, an experiment utilizing ptESI coupled with collision-induced unfolding is introduced, describing the application of the acquisition software and the subsequent programmatic analysis. Overall, this work marks a significant programmatic milestone towards automating the collection of both MS and IM-MS data utilizing the ptESI source. This work represents significant milestones towards being able to screen protein libraries or protein-ligand libraries in drug discovery and development.

Appendix A supplements Chapter 2 and contains a render of the physical ptESI source. Appendix B supplements Chapter 3 and contains examples of input and output data and visualizations related to the analytical pipeline. Appendix C supplements Chapter 4 and contains additional visualizations to assist with the characterization of ptESI, CIU, and ptESI coupled with CIU work.

Acknowledgements

I would like to thank those that have supported me throughout the years. My journey to graduate work has not been linear and this accomplishment is a major milestone in my life. First, I would like to thank those that have been in my life for a long time. I would like to thank my parents and my sister for continuing to believe in me despite major setbacks. I would like to acknowledge and thank Dr. Kiley Miller for his initial training and continued mentorship over the past decade. He put me in touch with a mentor, Tanya Jimenez, whom I would also like to thank. She has been a major source of support during my work as a data scientist and as I've pursued this degree. I would also like to thank Brooke Bauch for helping to keep my love and passion for analytical chemistry alive as we have both maneuvered through various roles since graduating from our undergraduate institution. I would also like to thank my paternal grandfather for making the completion of this degree possible and my maternal grandmother for modeling grit.

Secondly, I would like to thank those that are more recent additions to my life. The support of the 2023-2024/5 Applied Chemical Sciences and Technologies cohort has been crucial to my success today. I would like to thank Dr. Matthew F. Bush for his willingness to give me space in his research group to work towards my professional goal of empowering cool science by connecting people with academic pursuits through the medium of technology. Each member of the Bush Lab has contributed to my success in some small way, and I am thankful to each of you. I would like to specifically acknowledge Christopher J. Weir for laying the foundation of code that I contributed to and for many long hours of discussion regarding data structure. Also, I would like to acknowledge those that trained him, both Dr. Theresa Gozzo and

Dr. Meagan Gadzuk-Shea for their early work on the technology I worked on. I would also like to acknowledge those that trained me: May Constabel, Carley Hinrichs, Christopher Weir, Addison Roush, and those that trained with me Lilly Woerner and Anna Lin.

Lastly, I would like to thank my partner for his unyielding support.

Table of Contents

CHAPTER I

Introduction

This chapter introduces the need for protein stability measurements and how mass spectrometry may be used to achieve additional insights. Additionally, this chapter introduces a variety of electrospray ionization (ESI) techniques that augment solution temperature as an approach to uncover protein stability measurements. Subsequently, the idea of augmenting the temperature of the solution prior to the ESI process in tandem with utilizing ion mobility is introduced.

Figures of Merit for Protein Stability Measurements

One of the many challenges that faces the discovery and development of drugs used to treat medical conditions, such as recombinant proteins or monoclonal antibodies, is that they may be unstable or prone to aggregation. It is important to discover this limitation early in the research and development timeline so that new options may be onboarded while balancing the cost of development. Additionally, rapid screening is beneficial to further balance the time and financial costs of testing the stability of a promising product or suite of products. Protein unfolding and aggregation in terms of products can have an impact on quality, especially safety, efficacy, and immunogenicity.[1–3] Protein based drugs are sensitive to chemical and environmental factors such as temperature, shear-forces, light, pH, glycosylation, and enzymatic action.[4] Being able to probe for aggregation during manufacturing of protein-based drugs is essential for achieving a final product that is suited for long-term storage and reducing impurities that can cause and anti-drug immune response.[1,5,6]

Traditional Methods

Traditional techniques, X-ray crystallography, NMR spectroscopy, and cryo-electron microscopy (cryo-EM),[7–10] that have measured protein structure require a significant amount of pure and stable material, orders of magnitude greater than that that would be used in medication. Protein purification strategies used to acquire volumes high enough to be used in these traditional methods often cause the protein to misbehave. Subsequent analysis is, as such, no longer reflective of the structure of the protein when being used as a medication in a clinical setting.

Other techniques that have measured protein stability include differential scanning calorimetry (DSC), differential scanning fluorimetry (DSF), and circular dichroism (CD). DSC and DSF measure the protein midpoint transition in melting (T_m).[11] CD measures absorbance at specific wavelengths and, when monitored alongside a second experimental condition such as temperature, can yield information about secondary structure characteristics changes which results in a stability measurement.[12] Both DSC[11] and CD[12] require larger sample volumes. While DSF does not require larger sample volumes for analysis, it does require the addition of dye or detergent that may shift the stability of the protein away from that of its clinical application.[12]

Mass Spectrometry (MS)

Native mass spectrometry (MS) is a good fit for both structure and stability analysis, as the protein ions will retain a state similar to their solution phase conditions.[13,14] Another advantage of using MS is that the amount of protein sample required for nano-electrospray ionization (nESI) is less than a nanogram for several experiments. Ions for use in MS, a gas-phase technique, are first generated using an ESI source, which moves the prepared sample, using physiologically relevant pH and ionic strength, from a solution phase to the gas phase.[15]

MS can probe for detailed structural information by evaluating the transitions during molecular events such as changes in temperature.[16] For instance, if a changing temperature is applied to a prepared sample in the solution phase and an ESI source is used to migrate the prepared sample from the solution phase to the gas phase, then the heat-induced denaturation of proteins can be analyzed. This is accomplished by using MS to monitor the changes in the ions as the solution-phase temperature is changing.[17] When this technique, known collectively as temperature-controlled ESI (tcESI), is coupled with the evaluation of ion mobility (IM), a gas-phase structural technique, then additional structural information alongside stability information can be studied.[18]

Temperature-Controlled Electrospray Ionization (tcESI)

When utilizing tcESI techniques, protein structure can be evaluated by comparing the different charge state distributions at varying temperatures.[19] Using tcESI also allows for a stability measurement T_m when comparing the average charge state from the charge state distribution at each temperature to the temperature.[19] As such, tcESI is a powerful tool to evaluate both the structure and stability of proteins. This section describes a variety of tcESI techniques that are used in mass spectrometry including variable-temperature electrospray ionization (vtESI) and programmed-temperature electrospray ionization (ptESI).

Variable-Temperature Electrospray Ionization (vtESI)

Variable-temperature electrospray ionization techniques aim to garner both structure and stability information by analyzing mass to charge ratios (m/z) or a collision cross section, which is a rotationally averaged metric describing the size of an analyte. The vtESI system takes MS measurements at discrete temperature points and often only includes heating data and not cooling data. Additionally, many of the vtESI stability associated calculations require the use of a

software called OriginPro (OriginLab Corporation, Northampton, MA).[18,20,21] This additional software requirement makes this vtESI technology less accessible and less globally adoptable.

Programmed Temperature Electrospray Ionization (ptESI)

Past bodies of work have described a programmed-temperature ESI (ptESI) that address the limitation of requiring discrete temperature measurements during both heating and cooling, by employing continuous temperature control and mass spectra acquisition.[22,23] Rapid temperature cycling with high fidelity between the solution temperature and the programmed temperature has been made possible using the ptESI source.[22,23] The ptESI source was originally built in 2020 by Dr. Meagan Gadzuk-Shea who performed successful proof-of-principle experiments. The project was picked up again in 2023 by Dr. Theresa Gozzo, Christopher Weir, and May Constabel.

Temperature-Induced Unfolding (TIU)

Recently, vtESI has been coupled with collision-induced unfolding (CIU), a type of MS analysis that evaluates CCS as it relates to a collisional activation caused by applied voltages in a region of the mass spectrometry that is prior to the ion mobility region.[24] This study was able to uncover a compact conformer in Cytochrome C at elevated solution temperatures where most of the population unfolds.

Overview of Present Work

Although the hardware for the ptESI source has been described previously,[22,23] the programmatic control of said hardware has not been characterized. A high-level overview of how the temperature control device interfaces with both MS and the software is shown in Figure 1.1.

This body of work aims to address the limitations regarding the use of proprietary software for analysis of structure and stability information for protein melting point. Additionally, ptESI was coupled with CIU to investigate a potential pathway to studying relatively stable proteins that prove resistant to ptESI based analysis of the structure and stability of proteins. To showcase this, Ribonuclease A was selected for analysis to complement past reduction work,[22] and to add additional proteins to those previously analyzed.[23]

Ribonuclease A (RbA) is a relatively stable protein with 4 disulfide bonds in the monomeric structure, shown in Figure 1.2. These disulfide bonds can model the 4 disulfide bonds in the quaternary structure of immunoglobins (IgGs). IgGs are not only naturally occurring but also are one of the classes of protein-based drugs that are, at the time of writing, being heavily researched and developed. IgGs fall into the type of protein-based drug described previously as suffering from attempts to manufacture bulk quantities, which makes MS an ideal tool to analyze their structure and stability. Since the analytical tool to perform this analysis is also under development, using a more cost-effective model protein can also balance the cost and time development of the analytical tool will take in early investigations. If there is an impact to the structure and stability of RbA, those same experimental conditions may perturb the more complex IgGs.

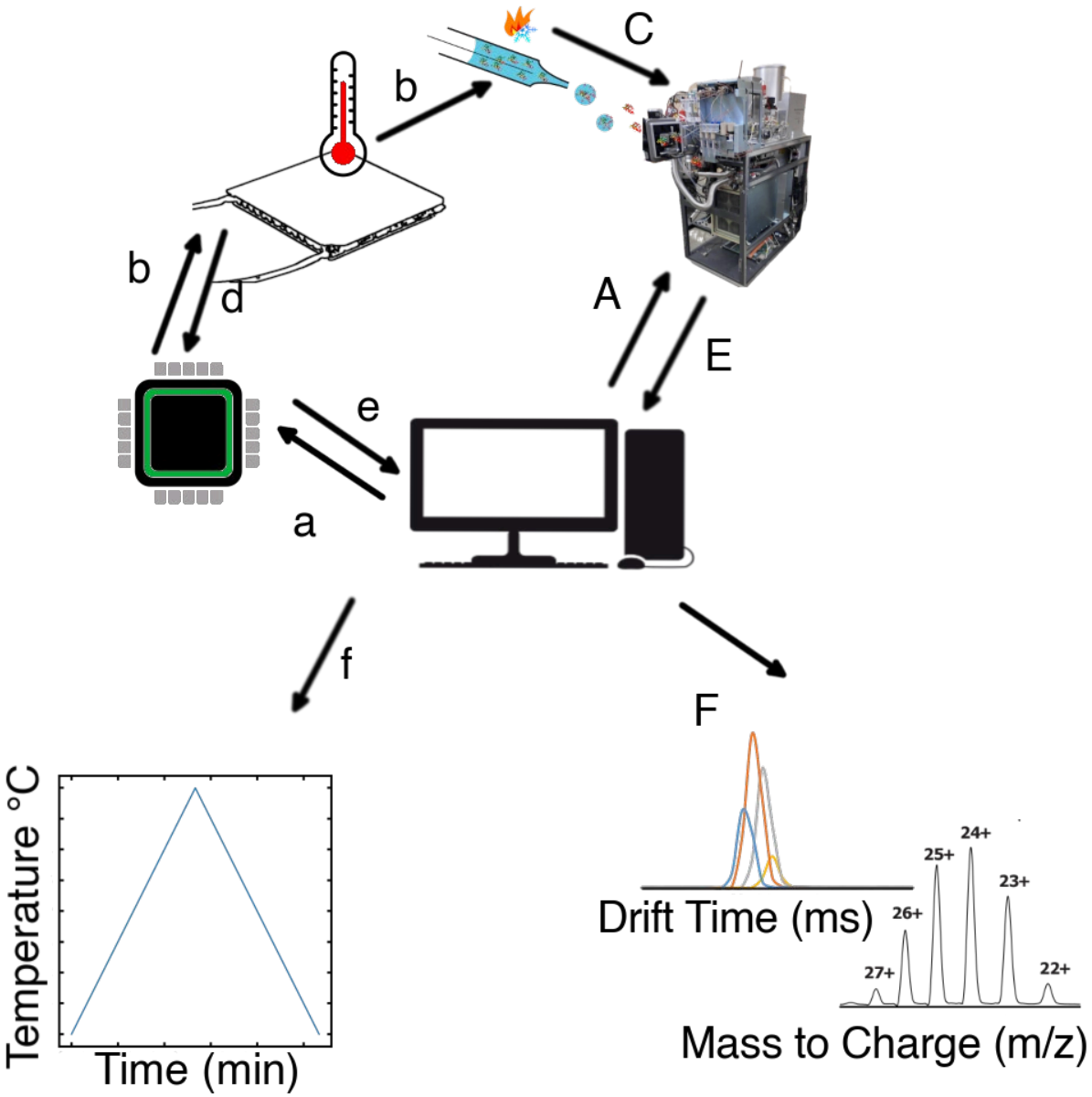


Figure 1.1. Ecosystem of acquisition where the uppercase letters correspond to mass spectrometry specific processes and where the lowercase letters correspond to the custom temperature controlled nESI specific processes that both occur on the same acquisition computer. Process A and process a represents that the acquisition computer starts both the MS acquisition program and the temperature acquisition program. Process b denotes that the microcontroller controls the bulk sample temperature. Process C represents the electrospray process and the start

of MS acquisition. Process d denotes the embedded thermistor readback to the microcontroller. Process E and process e represent that both MS data and temperature data are exported to the acquisition computer. Process F and process f represent the data that is exported from the acquisition computer to whatever system the user will analyze on where the temperature-controlled data and mass spectrometry data are aligned and processed for experimental insight into the structure of the protein.

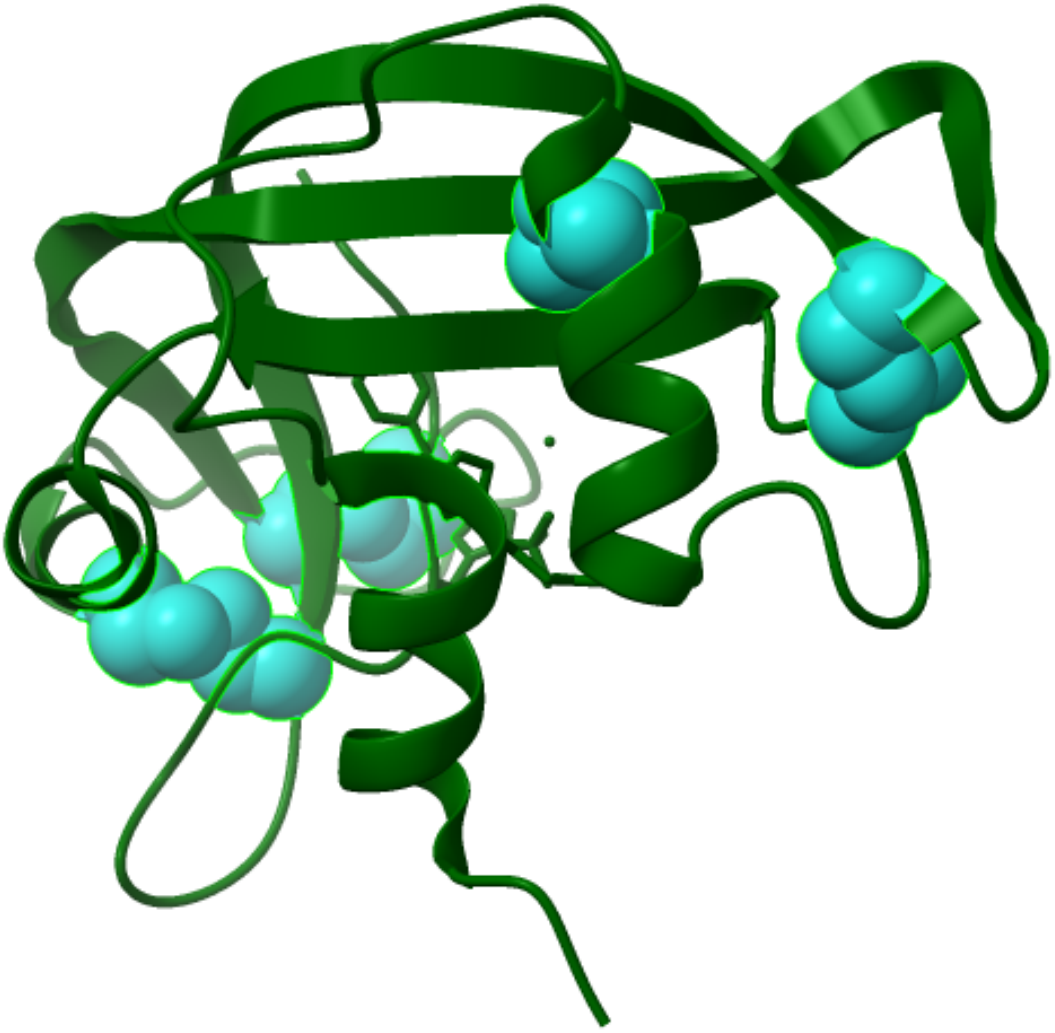


Figure 1.2. Ribonuclease A, 3RN3,[25–27] where the cysteine residues are shown in blue, and the remainder of the residues are green. At the bottom of the image is the N terminus and behind the central beta sheet, the C terminus is just visible. Of note is that there are three alpha helices and one beta sheet.

References

(1) Ratanji, K. D.; Derrick, J. P.; Dearman, R. J.; Kimber, I. Immunogenicity of Therapeutic Proteins: Influence of Aggregation. *J. Immunotoxicol.* **2014**, *11* (2), 99–109. https://doi.org/10.3109/1547691X.2013.821564.

(2) Roberts, C. J. Protein Aggregation and Its Impact on Product Quality. *Curr. Opin. Biotechnol.* **2014**, *30*, 211–217. https://doi.org/10.1016/j.copbio.2014.08.001.

(3) Krause, M. E.; Sahin, E. Chemical and Physical Instabilities in Manufacturing and Storage of Therapeutic Proteins. *Curr. Opin. Biotechnol.* **2019**, *60*, 159–167. https://doi.org/10.1016/j.copbio.2019.01.014.

(4) Rajan, R.; Ahmed, S.; Sharma, N.; Kumar, N.; Debas, A.; Matsumura, K. Review of the Current State of Protein Aggregation Inhibition from a Materials Chemistry Perspective: Special Focus on Polymeric Materials. *Mater. Adv.* **2021**, *2* (4), 1139–1176. https://doi.org/10.1039/D0MA00760A.

(5) Svilenov, H.; Markoja, U.; Winter, G. Isothermal Chemical Denaturation as a Complementary Tool to Overcome Limitations of Thermal Differential Scanning Fluorimetry in Predicting Physical Stability of Protein Formulations. *Eur. J. Pharm. Biopharm.* **2018**, *125*, 106–113. https://doi.org/10.1016/j.ejpb.2018.01.004.

(6) Ribeiro, R.; Abreu, T. R.; Silva, A. C.; Gonçalves, J.; Moreira, J. N. Insights on the Formulation of Recombinant Proteins. In *Current Applications of Pharmaceutical Biotechnology*; Silva, A. C., Moreira, J. N., Lobo, J. M. S., Almeida, H., Eds.; Advances in Biochemical Engineering/Biotechnology; Springer International Publishing: Cham, 2019; Vol. 171, pp 23–54. https://doi.org/10.1007/10_2019_119.

(7) Chari, A.; Haselbach, D.; Kirves, J.-M.; Ohmer, J.; Paknia, E.; Fischer, N.; Ganichkin, O.; Möller, V.; Frye, J. J.; Petzold, G.; Jarvis, M.; Tietzel, M.; Grimm, C.; Peters, J.-M.; Schulman, B. A.; Tittmann, K.; Markl, J.; Fischer, U.; Stark, H. ProteoPlex: Stability Optimization of Macromolecular Complexes by Sparse-Matrix Screening of Chemical Space. *Nat. Methods* **2015**, *12* (9), 859–865. https://doi.org/10.1038/nmeth.3493.

(8) Kozak, S.; Lercher, L.; Karanth, M. N.; Meijers, R.; Carlomagno, T.; Boivin, S. Optimization of Protein Samples for NMR Using Thermal Shift Assays. *J. Biomol. NMR* **2016**, *64* (4), 281–289. https://doi.org/10.1007/s10858-016-0027-z.

(9) Kotov, V.; Bartels, K.; Veith, K.; Josts, I.; Subhramanyam, U. K. T.; Günther, C.; Labahn, J.; Marlovits, T. C.; Moraes, I.; Tidow, H.; Löw, C.; Garcia-Alai, M. M. High-Throughput Stability Screening for Detergent-Solubilized Membrane Proteins. *Sci. Rep.* **2019**, *9* (1), 10379. https://doi.org/10.1038/s41598-019-46686-8.

(10) Kwan, T. O. C.; Reis, R.; Siligardi, G.; Hussain, R.; Cheruvara, H.; Moraes, I. Selection of Biophysical Methods for Characterisation of Membrane Proteins. *Int. J. Mol. Sci.* **2019**, *20* (10), 2605. https://doi.org/10.3390/ijms20102605.

(11) Lang, B. E.; Cole, K. D. Differential Scanning Calorimetry and Fluorimetry Measurements of Monoclonal Antibodies and Reference Proteins: Effect of Scanning Rate and Dye Selection. *Biotechnol. Prog.* **2017**, *33* (3), 677–686. https://doi.org/10.1002/btpr.2464.

(12) Stelea, S. D.; Pancoska, P.; Benight, A. S.; Keiderling, T. A. Thermal Unfolding of Ribonuclease A in Phosphate at Neutral pH: Deviations from the Two–State Model. *Protein Sci.* **2001**, *10* (5), 970–978. https://doi.org/10.1110/ps.47101.

(13) Gadzuk-Shea, M. M.; Bush, M. F. Effects of Charge State on the Structures of Serum Albumin Ions in the Gas Phase: Insights from Cation-to-Anion Proton-Transfer Reactions,

Ion Mobility, and Mass Spectrometry. *J. Phys. Chem. B* **2018**, *122* (43), 9947–9955. https://doi.org/10.1021/acs.jpcb.8b08427.

(14) Gabelica, V.; Shvartsburg, A. A.; Afonso, C.; Barran, P.; Benesch, J. L. P.; Bleiholder, C.; Bowers, M. T.; Bilbao, A.; Bush, M. F.; Campbell, J. L.; Campuzano, I. D. G.; Causon, T.; Clowers, B. H.; Creaser, C. S.; De Pauw, E.; Far, J.; Fernandez-Lima, F.; Fjeldsted, J. C.; Giles, K.; Groessl, M.; Hogan, C. J.; Hann, S.; Kim, H. I.; Kurulugama, R. T.; May, J. C.; McLean, J. A.; Pagel, K.; Richardson, K.; Ridgeway, M. E.; Rosu, F.; Sobott, F.; Thalassinos, K.; Valentine, S. J.; Wyttenbach, T. Recommendations for Reporting Ion Mobility Mass Spectrometry Measurements. *Mass Spectrom. Rev.* **2019**, *38* (3), 291–320. https://doi.org/10.1002/mas.21585.

(15) Leney, A. C.; Heck, A. J. R. Native Mass Spectrometry: What Is in the Name? *J. Am. Soc. Mass Spectrom.* **2017**, *28* (1), 5–13. https://doi.org/10.1007/s13361-016-1545-3.

(16) Wang, G.; Abzalimov, R. R.; Kaltashov, I. A. Direct Monitoring of Heat-Stressed Biopolymers with Temperature-Controlled Electrospray Ionization Mass Spectrometry. *Anal. Chem.* **2011**, *83* (8), 2870–2876. https://doi.org/10.1021/ac200441a.

(17) Mirza, U. A.; Cohen, S. L.; Chait, B. T. Heat-Induced Conformational Changes in Proteins Studied by Electrospray Ionization Mass Spectrometry. *Anal. Chem.* **1993**, *65* (1), 1–6. https://doi.org/10.1021/ac00049a003.

(18) El-Baba, T. J.; Woodall, D. W.; Raab, S. A.; Fuller, D. R.; Laganowsky, A.; Russell, D. H.; Clemmer, D. E. Melting Proteins: Evidence for Multiple Stable Structures upon Thermal Denaturation of Native Ubiquitin from Ion Mobility Spectrometry-Mass Spectrometry Measurements. *J. Am. Chem. Soc.* **2017**, *139* (18), 6306–6309. https://doi.org/10.1021/jacs.7b02774.

(19) Benesch, J. L. P.; Sobott, F.; Robinson, C. V. Thermal Dissociation of Multimeric Protein Complexes by Using Nanoelectrospray Mass Spectrometry. *Anal. Chem.* **2003**, *75* (10), 2208–2214. https://doi.org/10.1021/ac034132x.

(20) El-Baba, T. J.; Raab, S. A.; Buckley, R. P.; Brown, C. J.; Lutomski, C. A.; Henderson, L. W.; Woodall, D. W.; Shen, J.; Trinidad, J. C.; Niu, H.; Jarrold, M. F.; Russell, D. H.; Laganowsky, A.; Clemmer, D. E. Thermal Analysis of a Mixture of Ribosomal Proteins by vT-ESI-MS: Toward a Parallel Approach for Characterizing the *Stabilitome*. *Anal. Chem.* **2021**, *93* (24), 8484–8492. https://doi.org/10.1021/acs.analchem.1c00772.

(21) Brown, C. J.; Woodall, D. W.; El-Baba, T. J.; Clemmer, D. E. Characterizing Thermal Transitions of IgG with Mass Spectrometry. *J. Am. Soc. Mass Spectrom.* **2019**, *30* (11), 2438–2445. https://doi.org/10.1007/s13361-019-02292-6.

(22) Gozzo, T. A. New Ion Mobility-Mass Spectrometry Tools to Characterize and Differentiate Similar Proteins. PhD, University of Washington, 2023.

(23) Constabel, M. A. Programmed Temperature Electrospray Ionization (ptESI) for Thermal Cycling of Proteins. Master of Science, University of Washington, 2024.

(24) Jordan, J. S.; Chen, C. J.; Lee, K. J.; Williams, E. R. Temperature Induced Unfolding and Compaction of Cytochrome *c* in the Same Aqueous Solutions. *J. Am. Chem. Soc.* **2025**, *147* (4), 3412–3420. https://doi.org/10.1021/jacs.4c14267.

(25) Howlin, B.; Moss, D. S.; Harris, G. W.; Palmer, R. A. SEGMENTED ANISOTROPIC REFINEMENT OF BOVINE RIBONUCLEASE A BY THE APPLICATION OF THE RIGID-BODY TLS MODEL: 3rn3, 1991. https://doi.org/10.2210/pdb3rn3/pdb.

(26) Howlin, B.; Moss, D. S.; Harris, G. W. Segmented Anisotropic Refinement of Bovine Ribonuclease A by the Application of the Rigid-Body TLS Model. *Acta Crystallogr. A* **1989**, *45* (12), 851–861. https://doi.org/10.1107/S0108767389009177.

(27) Pettersen, E. F.; Goddard, T. D.; Huang, C. C.; Meng, E. C.; Couch, G. S.; Croll, T. I.; Morris, J. H.; Ferrin, T. E. UCSF ChimeraX : Structure Visualization for Researchers, Educators, and Developers. *Protein Sci.* **2021**, *30* (1), 70–82. https://doi.org/10.1002/pro.3943.

CHAPTER II

Software for Acquisition – ptESI-Control

Introduction

This chapter describes the approach in developing a new user interface for the software associated with the control of the programmed-temperature electrospray ionization (ptESI) source. The introduction discusses software definitions, previous methods utilized and their unmet needs, and the objectives for the new acquisition method. Next, the methods employed throughout the development of the new software are described. This is followed by information about the results of the software development process. Finally, there are concluding remarks.

Overall, the data acquisition process adheres to the design principles of the mzML format[1] and guidelines set forth by the HUPO PSI.[2] Data is acquired using a microcontroller programmed using customized version-controlled Python code. This encoded data is sent to the acquisition computer to be decoded and parsed into an accessory file designed to be a companion file to the mass spectra acquisition files.

Software Definitions

This paper will use the following definitions. A user is defined as someone who will access and use the software. A developer is defined as someone who interacts with the code base with the intention of making modifications. A virtual environment is the programmatic space in which Python and associated code is allowed to run on a computer. A class is defined as a set of actors within the code base that perform or execute actions. An instance is defined as the code base as it exists during the execution of the acquisition software for a single acquisition, after

which point the instance is dismissed and is no longer accessible. An object is defined as a single instance of a class. Methods are defined as the functions within classes which perform actions. Attributes are defined as the information associated with a specific class. Operations are defined as commands that allow for interaction with the code base, such as mathematical calculations or logical comparisons. Inheritance is defined as the ability to access the attributes of a class or object from a connected class in the fashion that the child class or object will inherit attributes from a parent class or object. A module is defined as a file containing Python definitions and statements. A library is a collection of modules. A package is a specific way to organize modules that utilizes an __init__.py file. The phrase package will be used in reference to the installable that is added to a virtual environment. The phrase library will be used when referencing the code contained within the installable package. A refactor is a process that revises and reformats existing code, sometimes optimizing prior algorithms, to align with new or planned systems and features.

Overview of Previous Methods for ptESI

This section reviews the previous software methods used to acquire ptESI data including using a command line interface and a procedural code base. Afterwards, a brief description of the unmet needs when utilizing this system is discussed regarding ease of use and ease of future updates.

Command Line Interface

Previously, data was collected using a command line interface (CLI) where an end user would start a windows command prompt, also known as a terminal, and manually type in commands for ptESI_Program.py access after commanding Anaconda, a virtual environment manager through which Python runs, to activate and navigating to the correct folder where the

code allowing programmed temperature control is saved. To increase accessibility and lower the barrier to entry for those with less coding background a choice was made to include a graphical user interface (GUI) in lieu of a CLI.[3]

Procedural Code Base

The earliest release of the code, the alpha release, [4] was created by a fellow Bush Lab member, Christopher J. Weir based on early work done by Dr. Meagan M. Gadzuk-Shea, a former Bush Lab member, and was packaged for release by myself after troubleshooting. It uses a CLI, and the programming style is distinctly procedural where one line of code happens after the next line of code and the entire code base proceeds line by line. This style of coding, while operationally works and often proceeds without problems, can increase the difficulty in code maintenance, scalability, and can have some security concerns.[5,6]

Unmet Needs

To empower access of the user to the software, the GUI uses a web framework, the Flask module, an imported Python library, where all commands, inputs, and outputs are on a single page to empower automation for high throughput analysis in the future.

To make the codebase easier to maintain and update, an object-oriented programming style has been developed and adopted. Additionally, this new approach allows for XML based logging of code events that can be exported as metadata[2], which allows for easier event tracking and will empower the development of future features.

Objectives for New Acquisition Method

This section discusses object-oriented programming and graphical user interfaces and how the use of both was employed for the development of the new acquisition software.

Objected-Oriented Programming (OOP)

Objected-Oriented Programming (OOP) is a method of programming that instead of proceeding in a specific order of events, as in procedural programming, organizes code by storing information inside of classes that have properties and are then able to do actions, in coding terms, having attribution and to executing methods, based on the specific contents, whether attributes or methods, of that class. This allows for different classes to be called into action at different moments within a codebase. Something of note is that in OOP, code happens on an instance basis. This means that when the code is started that the classes are instantiated, or turned on, to create an object. In order for the code to proceed with the different stored attributes and methods, the object must be referenced, this reference point is called an instance. The instance is disbanded at the end of code execution.

This structure adds a layer of security into the code base for ptESI-Control that was not present previously. Some class attributes are designed to be static, for example the microcontroller unit can only receive calls at a specific interval, called the sampling rate. This value can be set within the class prior to running the code and is inaccessible without creating an object instance and even then cannot be changed while the code is running. There are other attributes that are designed to be dynamic and updatable while the code is running, for example the temperatures that the user wants to investigate. However, this particular attribute is stored within the simulation class. Another class, for example the microcontroller class, can call on and read the attribute for temperatures, but cannot modify the attribute. This separation of variables and classes access to their own and other variables is called encapsulation.

Encapsulation minimizes interdependencies between modules. In object-oriented programming a module is defined as an object. An object is an instance of a class. In this

particular project, there are a variety of classes that include, but are not limited to, ports, simulation, and microcontroller. In this case encapsulation means that a different temperature microcontroller can be onboarded without impact on the port connections or the simulation running effectively. These different objects are essentially self-contained and are pooled together through running the adjacent Python code. Additionally, implementation of each class and object instance is hidden from outside classes or object instances. This protects the internal attributes and functions of each class or object instance, and limits troubleshooting to within a specific class rather than an entire code base.

Abstraction has been described as selective emphasis on detail by emphasizing what is significant to the user and suppressing what is not[7]. For this programmed temperature control project, it is important to know that there is a significant datetime component, that there is port handling, that there is a microcontroller and the specific type of microcontroller, there is the generation of a data file, that a thermocouple can be used alongside the embedded thermistor for validation of the temperature control, that there is a simulation that can be accessed, that multiple cycles can be instituted, and that the programmatic control is enforced using various counters. Abstraction also allows for the description of the behavior of an object without referencing direct representation. For example, an object instance attribute can be noted to be a list without the program actively seeing the variable containing the list.

OOP essentially allows for a template to be created that is only filled in at the moment of code execution, and while the template remains afterwards, the filled in portion is unique to that one moment. This idea is in alignment with what having unique experiments where the data is limited to that one singular acquisition.

Graphical User Interface

The abstraction that OOP created for this project is further embodied by the graphical user interface. This interface system moves the code under the hood so to speak, so that users only see the information that is necessary to execute the experiment removing mental load. This meets the demand of ease of use. Instead of needing to understand the variables present inside of a parameters file that has both changeable and unchangeable variables contained within it, the user only needs to answer the questions on the screen and verify that the simulation outputs the type of temperature profile that they want to implement and then press a series of buttons to control the microcontroller instead of typing code to execute a Python script after navigating to the correct directory and activating the virtual environment.

Methods

Python Libraries

A combination of Python libraries were needed to support the development of the acquisition software. Included are Python standard libraries, meaning that the library is pre-installed when using Python, but needs to be declared to become active in the code space, and imported Python libraries, which are not managed by Python directly, but can work in concert with Python.

Python Standard Libraries

In order to support the Python code base a series of previously developed packages are implemented in this project. The particular Python version for this project is Python 3.8.17 due to a restriction of using the latest version of the Python package pyserial, 3.5, which allows for data

transmission through a serial cable from the microcontroller to the acquisition computer.[8,9] The requisite data transmission is not supported in later versions of Python.

There are a series of packages that are imported to support the code workflow that are considered as standard and as such are built in with Python. This means that no additional installation to the virtual environment is necessary. The os package allows interactions with the operating system, and in this project, very specifically allowing access to both navigate to project specific folders through code execution and to generate and save the output files during code execution. The datetime package allows for the manipulation of datetime data, such as recording the moment in time that a specific temperature is read and recorded. The datetime library is foundational for the real time control of what temperature the microcontroller is set to reach. In earlier versions of the code that required the command line interface to operate, interaction with a command line from the Python code was required to do things like read out the current temperature and the elapsed time for the ongoing experiment. This particular package, argparse, is depreciated and removed from the virtual environment for the GUI version of the project. The time package is crucial for the programmatic operation of the microcontroller. For our use case, the time package allows the code, while it is in the process of executing, to “sleep” or pause for a period of time. This prevents the microcontroller from being overwhelmed by requests and keeps the code from missing sending specific temperatures to reach, meaning that every identified necessary temperature is actually sent to the microcontroller.

Imported

There are several Python packages that do need to be installed into the virtual environment for the operation of the code. The first of which is specific to command line releases of the code and is not included with the GUI version of the code. This package is

colorama which allows for more succinct viewing of output information to the command line.[10] While it is not necessary for code function, it does make the command line version of the project more accessible for use. Instead, the GUI version of the project requires the installation of the Flask and werkzeug packages.[11,12] Flask is a Python-specific web-server gateway interface (WSGI) web framework that the backend of the GUI is built on and it requires werkzeug as a dependency. Werkzeug is a WSGI web application library that handles responses from the environ, in other words, utilizing this Python library encodes and decodes information that is sent from page to page within the application while adhering to PEP 3333.

Although these packages discuss viewing the application, there are packages that allow for data to be handled and transmitted with the application appropriately. One such package is NumPy which allows scientific data management and manipulation.[13] Another package that the ptESI-Control project requires allows for the creation of graphs and the visualization that empowers interpretation called Matplotlib.[14]

Additional Python packages enable Python to interact with a microcontroller, in this specific project a temperature microcontroller. The first package is pySerial, previously mentioned to the limiting factor in what version of Python can be used for this project. PySerial is a package that allows Python to connect to a serial port connection.[9] This creates the bridge between the microcontroller and the acquisition computer allowing for data transmission and capture, which is vital to the success of this project. Additionally, a package called mcculw, short for measurement computing universal library for windows, allows for interactions with the input output products in the Universal Library.[15] In this project the input output product that is used is the thermocouple that is used to ensure temperature integrity between the programmed temperature and the readout temperature. This package is necessary to get the temperature

readout from the thermocouple device when it is connected to the acquisition computer and in use alongside the ptESI-Control program.

Organization – Repository

This acquisition software is designed to be an all-in-one package that is installable. The layout of the package is showcased in Figure 2.1 which describes the contents of the code containing repository. Data will populate in the data folder throughout the course of the acquisition. The backend of the code for the GUI is contained in the blueprints folder, specifically within the pages.py file. The frontend of the code is contained in the static and templates folders. The code is launched using the configuration provided in the config.py and the __init__.py files which merges the backend and frontend of the code together to provide the functional GUI. The core_util.py file contains the OOP that is referenced throughout the GUI code. The core_ptESI_control.py file is not referenced in the GUI but is still usable as a backup and reference point for the order of operations in the GUI, however it utilizes a CLI. The setup.py file contains the metadata associated with the acquisition software. The reports folder contains the records of acquisition. The references folder contains a guide on using the associated hardware. The venv folder, environment.yml, requirements.txt, and .gitignore files all contain information about the code space that is utilized by the acquisition software. The LICENSE file contains the intellectual property information associated with the acquisition software. The README.md file contains information about the acquisition software, specifically some other open-source software that is necessary for the full operation of the acquisition software.

```
ptESI-Control/
|-- data/
    |-- interim/
        |-- parameters.xml
        |-- simulation.xml
    |--[date]_tempdata_[file_tag].xml
|-- docs/
|-- notebooks/
    |-- tempprog_examplerun.ipynb
|-- ptESI_control/ (src)
    |-- blueprints/
        |-- pages.py
    |-- static/
        |-- uploads/
        |-- manifest.json
        |-- ptESI_control_manifest.png
        |-- ptESI_control_icon.png
        |-- style.css
    |-- templates/
        |-- errors
            |-- 404.html
            |-- 500.html
        |-- pages/
            |-- home.html
            |-- parameters.html
            |-- run.html
            |-- simulation.html
        |-- base.html
    |-- __init__.py
    |-- config.py
    |-- core_ptESI_control.py
    |-- core_util.py
    |-- ptESI_control_icon.png
|-- references/
    |-- How to use the ptESI Source.pdf
|-- reports/
    |-- figures/
        |-- temp_prog_sim.png
    |-- code_events.log
    |-- figures.mplstyle
|-- venv/
|-- .gitignore
|-- environemnt.yml
|-- LICENSE
|-- README.md
|-- requirements.txt
|-- setup.py
```

Figure 2.1. The repository organization for the GUI version of the project.

Access to Software

Contact the Bush Lab for access to the ptESI-Control repository.

Results and Discussion

This section discusses the design work on the code base and the acquisition software generated outputs. The ptESI-Control library was designed utilizing OOP and a GUI specifically in a single-page web-framework real-time application factory. The single-page aspect was achieved using templates. The web framework leveraged was Python Flask. The real-time features were enabled by the utilization of decorators and an OOP code structure. The application factory was implemented through the use of the setup.py and __init__.py files alongside the use of blueprints. Additional functionality to meet the unmet needs of the previous code base include automatic port detection and a specific naming convention for class related functions.

Design of ptESI-Control Library

This subsection discusses OOP implementation, with an in-depth discussion of composition, and GUI creation, especially regarding templates and blueprints and their use. Also discussed is the use of decorators and the function naming scheme developed to work with the identified actors and their associated actions. Subsequently the added feature of automatic port detection to support the ease-of-use goal is discussed. This section wraps up with a description of the developed custom code bases for both the OOP and GUI implementations of the acquisition software.

Object-Oriented Programming

To meet the unmet need of easily updatable, the procedural code needed to be adapted to object oriented programming. To achieve this, I started by identifying the different phases of the code base. Ultimately there are three different phases throughout normal operation of the microcontroller, an input phase, a processing phase, and an output phase, as seen in Table 2.1.

After identifying this major component of functionality, I made a list of all the functions, the actions, in the code and adapted many operations to be contained within a function, so that they would be included. I then evaluated the list of functions to identify how data was being transmitted, whether it was from the computer to the microcontroller or the computer to the computer, or any other combination.

Once I had a comprehensive list of data transmission types, I identified the actors, in Table 2.11, that were engaging in those data transmission steps. Those actors became the building blocks of my object-oriented code. After identifying the actors and their behaviors, I updated the function naming scheme to be representative of the behavior of those identified actors, Table 2.1.

These steps allowed me to identify the number and type of classes that would need to be built out, and how different classes may need to interact with each other. This interaction piece informed the class that functions were migrated into becoming those attributes of the class. After the classes and attributes were generated, I was able to identify what information needed to be saved to the class indefinitely and what information needed to be updated by inputs from someone trying to run a temperature program. These static and dynamic attributes were assigned within the class structure.

Composition

Composition was embedded at this design point in the migration from procedural code to object-oriented code. Composition is the grouping of ideas, concepts, or actors doing behaviors that create something new. For example, the user input parameters with the wave function generator function create something new, a simulation. Although this simulation step was embedded in all previous versions of the ptESI-Control code and not separated out within the code base. I wanted to encapsulate and protect this aspect of the code base specifically by carefully designing the class with abstraction and privatization in mind. Essentially, this creates a read-only version of the user parameters input and then hides the specific variable information. You can see that there is a wave function type selected, but not which specific wave function, without the appropriate permissions within an instance of the code.

This was achieved by creating the class attributes to be protected, for example self._wave_type within the SimulationMeta class. These values were then passed through inheritance to the Simulation class to get information about the specific values contained within an instance. The Simulation class only gives access to the lengths of the submitted lists which is also protected.

Through creating composition, another class leveraged inheritance, the TC720 class. This class was created for the model of the microcontroller currently in use and is set up to inherit from the Microcontroller class. This allows for polymorphism. Polymorphism is the ability to have different handling within a class based on the subclass that is active. This means that a multitude of microcontrollers can have their own subclass within the Microcontroller class, each with their own independent Python commands for controlling the solution temperature, without needing to update the other classes for the function of the ptESI-Control module. For example, if

we onboard a new microcontroller from a different manufacturer, we can follow the suggestions of the manufacturer for Python control in a new microcontroller subclass, but the Simulation class will function similarly agnostically of this update to hardware due to the newly introduced polymorphism in the object-oriented programming version of the code. This phenomenon is known as duck typing. If a microcontroller can control temperature, this object-oriented version of the code will let it do that while also making the simulation and intaking user identified parameters input.

As a note, this code was developed with doc strings that contain information about the concept, the code logic, or the function of each code element. The doc string for each function also contains information surrounding the involved actors, their behavior and the requisite inputs and what the function will return.

Graphical User Interface

After the object-oriented version of the code was released, some beta testing was completed and a few bugs surrounding documentation of the temperature during the run were corrected. Finally, work on the graphical user interface began. To start, I gave a talk discussing both Flask and Django as potential options for the web framework for the graphical user interface. After discussion with current users of the command line interface and keeping in mind the early identified tenant of ease of update, I opted to endorse Flask, and the group reached a consensus. I hosted a short workshop for current users of the command line interface and guided them through some questions that led to everyone drawing out what they think the best version of the ptESI-Control application could be. I identified some common themes in the differing ideas and moved forward with those ideals in mind. I opted to move forward with single page web framework real time application factory made by Python Flask.

Since Flask uses single requests to perform its operations, meaning one input yields one singular output, the object-oriented execution of the code needed to be updated significantly. I identified the major code blocks within the object-oriented version of the code. In this case code blocks were identified as being mostly self-contained with respect to the required inputs and outputs. The major code blocks identified include setting the parameters, executing the simulation, and completing the run itself. Having these specific allocated code blocks allowed for the creation of web pages that specifically focus on doing only that single action. In other words, each code block has a specific input and output, meaning that Python Flask can now process requests made to the application and render the information for viewing through the lens of the frontend HTML, CSS, and JavaScript (JS) code. The Python code files specifically interact with the classes and then Flask encodes the generated outputs for viewing

Templates – Supporting the Rendering of Visual Information

These three units, parameters, simulation, and run, have separate html pages with embedded JS for Python output processing that are fed into the singular home page. This means that all three pages can be seen on the home page, fulfilling the single-page application request by group members. This is largely possible due to Asynchronous JavaScript and XML (AJAX) that allows for the display of html encoded information without a full page reload, in this projects case, by filling in a predefined html template.

For this project there is (1) a base template, which provides the general outline for all of the pages, and contains most of the visual layout properties, (2) a parameters template, which sets up the interface to receive parameter input information, (3) a simulation template, which set ups the interface to munge, or transform the inputs into new outputs through data transformations and processing, (4) a run template, which provides buttons for user control of the run along with

status indicators, and (5) a home template, which incorporates the parameters, simulation, and run templates, providing the overall interface for the project. Although this requires the management of more files, the separate units can be updated independently of each other, introducing additional modularity.

The JS associated with each template is embedded within instead of broken out into separate files. While being in separate files would support modularity and easier version control, it would add communication overhead, potentially preventing the real-time component required by this project. The status and progress updates for the run template would suffer.

Blueprints – Supporting the Handling of the Application

The blueprint method of creating a Flask application allows for many instances of the same application to run. This decision was made, one for scalability, but also to ensure that every time the ptESI-Control module is run that it is run in the same fashion, and to prevent information from one run transferring into another run. Each time the application is started, it is with fresh input requests. Whenever the application is closed out, the instance and information stored in objects is dismissed. This prevents sticky parameters. The implementation of a blueprint system uses html templates and changes the structure of the application creation to be inside of a function instead of procedurally generated. This adoption is in alignment with the goals guiding the previous migration to object-oriented programming.

Table 2.1. Object oriented actors and their behaviors according to the phase of the code base. Of particular note is that the ports object has no processing behavior and that the microcontroller has no outputting behaviors and is the only object with the send behavior in the processing portion of the code base.

		Phases		
		Input	Processing	Output
Actors	Date Time	Connect	Transform Retrieve	
	Ports	Connect Configure		Close
	Microcontroller	Configure	Transform Retrieve Send	
	Data File	Connect Configure	Transform	Write Close
	Thermocouple	Connect	Retrieve	Close
	Simulation	Connect	Transform	Write
	Cycle		Retrieve	
	Counter	Configure		

Decorators

A coding specific that allows the protection of parameters input information and empowers templates is the use of something called a Python decorator, which is denoted by an "@" symbol in the code base. For the protection of the parameters input information, both property and setter decorators are used. To access the information contained within both properties and setters specific access through the instantiated object is required.

Templates are enabled through route decorators which contain the url of the page of the application. The functions underneath that decorator only pertain to that page. It is for this reason that all of the Python functions for the distinct html templates are contained within the one

pages.py file. The route decorators allow for direct correlation with the associated html rendering. There is one other type of decorator used by flask, the after this request decorator, which allows for the real-time handling of information from Python munging to displaying the information.

Functions

Both the classes and attributes in the utility Python file are organized in the order that they appear in the procedural code. The classes are named for the actors. The attributes are named according to the behavior contained within. Attributes start with only the following words: connect, configure, transform, retrieve, send, display, write, and close. Display after the original object-oriented programming code was developed to add necessary support for GUI specific rendering of graphical information. The additional words in the attribute naming scheme include what the behavior is performed on if not the object itself. For example, the transformStartTemperature attribute contained within the simulation class takes the user input information from the parameters and extracts and returns the first temperature from within the list of temperatures provided by the user. Another example contained within the datafile class is the attribute writeInformationToFile. This attribute records the collected information to the run output file.

Addition of Automatic Port Detection

Automatic port detection was deemed necessary to support the ease of use tenet for this project. In previous versions of the project, the port information was included with the parameters input file and required users to identify the port that the temperature controller connected to themselves. One of the goals of this project was to make the software more “plug and play” without the need for additional configuration. This required a programmatic way to

identify the connected port every time the temperature controller was plugged in. I added this feature by leveraging the included check for the model code for the specific model of microcontroller that is currently in use. The object-oriented programming nature of the software allows for additional subclasses under the class of microcontroller to be added to add specifics for different models of temperature controllers. The code that showcases this automatic port detection feature is in Figure 2.2.

```
def connectCommunicationFromMicrocontroller(self):
     potential_ports = [f'com{i}' for i in np.arange(0,257,1)]
     ...
     for i in range(0, len(potential_ports)):
          try:
               ser = serial.Serial(port=potential_ports[i],
                                   baudrate=230400, timeout=1)
               ...
               check_models.append(TC720.configureSerialCommand(
                                   TC720, ser,
                                   TC720.tc720_check))
               ...
          except serial.serialutil.SerialException as err:
               check_models.append([b'*',
                         b'X', b'X', b'X', b'X', b'6', b'0',
                         b'^', 0, 0, 0, 0, 0])
     return potential_ports, screened_port_indexes, check_models
```

Figure 2.2. Code excerpt showcasing automatic port detection

Custom Code Base

Object-Oriented Programming

For the OOP code base there are three primary files that communicate during the execution of the code, the core ptESI control Python file, the core Python utility file, and the Python parameters files. Both the contents of the core utility file and the parameters file are imported into the core ptESI control file prior to code execution. The core ptESI control file cannot be run independently of these two files, requiring the contents of the files.

Before the implementation of the OOP code base the only functions that were separated from the ptESI program Python file were the wave generation functions which have been migrated into the simulation class in the OOP code in a specific wave generation Python file.

Graphical User Interface

The core ptESI control Python file is deprecated for the GUI code base but is included, and was used, as a reference for the specific order of operations happening within the ptESI-Control module. The parameters Python file has also been deprecated, the contents of which guided the structure of the parameters related components of the GUI. In both the procedural code and OOP code the parameters file was used as an input and was stored alongside the data output for a record of the run constraints. In the GUI, parameter options are chosen or written in within the application interface and an xml file with the parameters values is exported alongside the run output to retain a record of the run constraints. The remaining utility file is still in use with some GUI support additions and updates within the Python file.

The full description of file communication while the application is running is illustrated in Figure 2.3. The __init__.py file builds the application by calling in all the necessary components and setting some defaults like which file types are allowed to be read in and created by the application. The create app function within the __init__.py file is used by the pages.py file to start blueprint generation. The blueprints, contained in both the html and display sections in Figure 2.3, which are written in Python directly communicate with the templates to render the code content into a graphical output. The templates contain embedded JS with enable buttons to both pass information to and call on the blueprint contents to preform Python programmatic processes, in other words the blueprints interface with the classes and allow for their instantiation

and enable access to their attributes, and are contained in the Python section in Figure 2.3. The templates also provide real-time readouts for the graphical display using embedded JS.

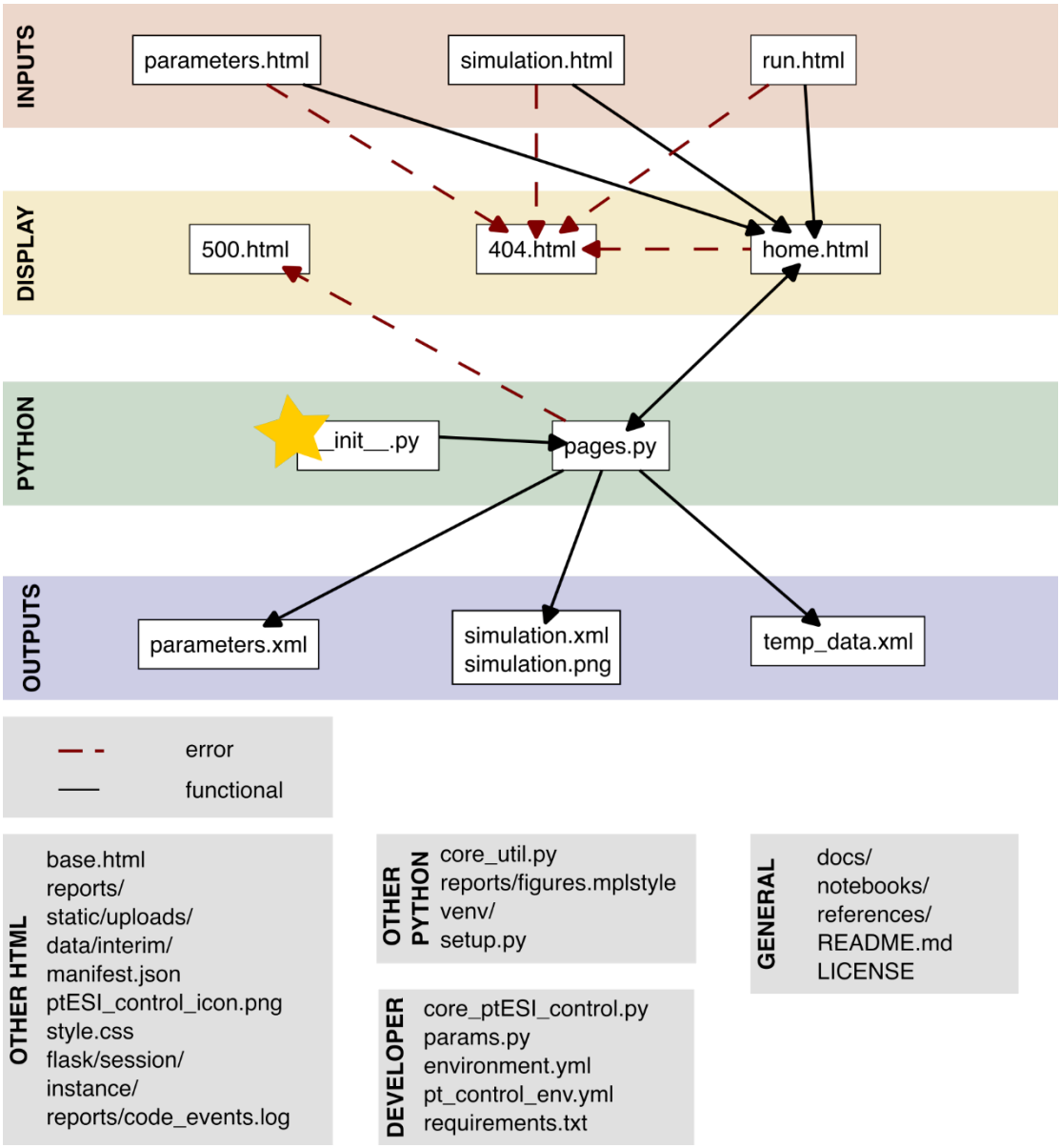


Figure 2.3. Pathway of inputs and outputs within the three primary functions of the run parameters generation, simulation of said parameters, and then finally running the experiment based on the simulation.

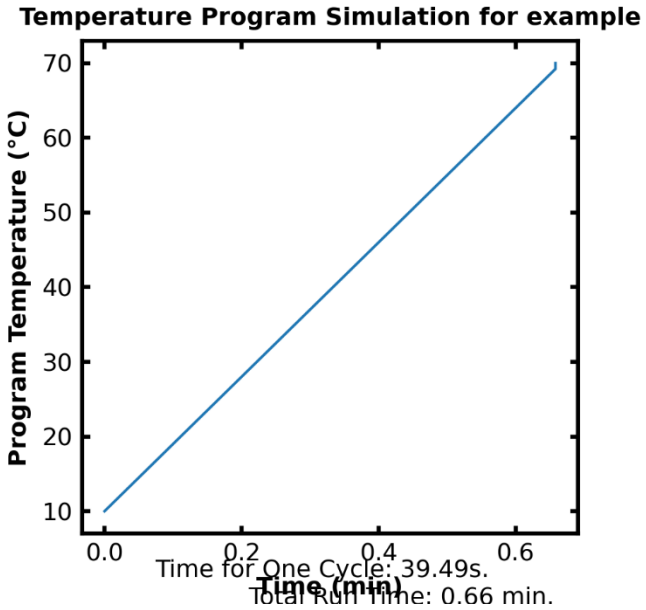
Design of Acquisition Log

The acquisition logs are exported in XML to be in compliance with the Human Proteome Organization (HUPO) Minimum Information About a Proteomics Experiment for Mass Spectrometry (MIAPE-MS) describes that tools involving data should use XML based systems to transport information[2]. As such, the parameters, simulation, and run information generate XML export files after they complete their function.

The parameters file exports the user input information, as seen in Figure 2.4. The simulation file exports the design of the experiment based on the parameters, as seen in Figure 2.5. The run file exports the information surrounding the microcontroller including the elapsed time, the current temperature, the set temperature from the simulation, the power draw, the current draw, the heat reservoir temperature, and the current time, as seen in Figure 2.6.

```
<?xml version='1.0' encoding='utf-8'?>
<parameters>
    <file_tag>example</file_tag>
    <inc_list>[0.0]</inc_list>
    <microcontroller>TC720</microcontroller>
    <mirror>1</mirror>
    <num_cycles>1</num_cycles>
    <rate_list>[1.5]</rate_list>
    <safe_temp>25.0</safe_temp>
    <temp_list>[10.0, 30.0, 50.0, 70.0]</temp_list>
    <temp_logger>False</temp_logger>
    <wave_type>0</wave_type>
</parameters>
```

Figure 2.4. Output of the parameters portion of the GUI, parameters_[file_tag].xml




```
<?xml version='1.0' encoding='utf-8'?>
<simulation>
    <sim_send_times><time_0>0.0</time_0>...</sim_send_times>
    <sim_send_temps><temp_0>10.0</temp_0>...</sim_send_temps>
</simulation>
```

Figure 2.5. A) Output of the simulation section of the GUI, temp_prog_for_[file-tag].png B)

Output of the simulation section of the GUI, times_temps_simulation_for_[file-tag].xml.

```
<?xml version='1.0' encoding='utf-8'?>
<run>
    <time_0>
        <time_elapsed></time_elapsed>
        <set_temp></set_temp>
        <sim_send_time></sim_send_time>
        <read_temp></read_temp>
        <logger_temp></logger_temp>
        <power_level></power_level>
        <current_output></current_output>
        <reservoir_temp></reservoir_temp>
        <current_time></current_time>
    </time_0>
    <time_1>
        ...
    </time_1>
</run>
```

Figure 2.6. Output of run portion of GUI, CCYY-MM-DD_tempdata_[file-tag].xml

Conclusions

For this project, to meet the need for ease of updates, an object-oriented programming approach was adopted. After this stage of the project was completed, the graphical user interface was implemented to meet the need for ease of use. Additionally, there was an update to how the outputs were generated to allow for data stability over time and integration with other developed modules. The results of this work can be seen in the v.0.1.0-alpha release as the procedural code. The v.0.2.1-beta release showcases the migration to OOP. The v-0.3.0-beta release contains the adoption of a graphical user interface. The releases can be found in the Bush Lab organization on GitHub.

Acknowledgements

This work was supported by the National Science Foundation through award 2203513 (Bush) from the Division of Chemistry, with partial co-funding from the Division of Molecular and Cellular Biosciences.

References

(1) Deutsch, E. W. File Formats Commonly Used in Mass Spectrometry Proteomics. *Mol. Cell. Proteomics* **2012**, *11* (12), 1612–1621. https://doi.org/10.1074/mcp.R112.019695.

(2) Taylor, C. F.; Paton, N. W.; Lilley, K. S.; Binz, P.-A.; Julian, R. K.; Jones, A. R.; Zhu, W.; Apweiler, R.; Aebersold, R.; Deutsch, E. W.; Dunn, M. J.; Heck, A. J. R.; Leitner, A.; Macht, M.; Mann, M.; Martens, L.; Neubert, T. A.; Patterson, S. D.; Ping, P.; Seymour, S. L.; Souda, P.; Tsugita, A.; Vandekerckhove, J.; Vondriska, T. M.; Whitelegge, J. P.; Wilkins, M. R.; Xenarios, I.; Yates, J. R.; Hermjakob, H. The Minimum Information about

a Proteomics Experiment (MIAPE). *Nat. Biotechnol.* **2007**, *25* (8), 887–893. https://doi.org/10.1038/nbt1329.

(3) Stephanidis, C. *The Universal Access Handbook*; Human Factors and Ergonomics; Taylor and Francis: Hoboken, 2014.

(4) Weir, C. Alpha CLI for ptESI-Control, 2024. https://github.com/bushgroup/ptESI-Control/releases/tag/v0.1.0-alpha.

(5) Lanning, D. L.; Khoshgoftaar, T. M. Modeling the Relationship between Source Code Complexity and Maintenance Difficulty. *Computer* **1994**, *27* (9), 35–40. https://doi.org/10.1109/2.312036.

(6) Bertino, E. Data Security and Privacy: Concepts, Approaches, and Research Directions. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*; IEEE: Atlanta, GA, USA, 2016; pp 400–407. https://doi.org/10.1109/COMPSAC.2016.89.

(7) Shaw, M. Abstraction Techniques in Modern Programming Languages. *IEEE Softw.* **1984**, *1* (4), 10–26. https://doi.org/10.1109/MS.1984.229453.

(8) Python. https://docs.python.org/release/3.8.17/.

(9) Liechti, C. pySerial. https://pythonhosted.org/pyserial/.

(10) Hartley, J.; Yaari, A. Colorama. https://pypi.org/project/colorama/.

(11) Flask. https://flask.palletsprojects.com/en/stable/.

(12) Werkzeug. https://werkzeug.palletsprojects.com/en/stable/.

(13) Harris, C. R.; Millman, K. J.; Van Der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; Van Kerkwijk, M. H.; Brett, M.; Haldane, A.; Del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant,

T. E. Array Programming with NumPy. *Nature* **2020**, *585* (7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2.

(14) Caswell, T. A.; De Andrade, E. S.; Lee, A.; Droettboom, M.; Hoffmann, T.; Klymak, J.; Hunter, J.; Firing, E.; Stansby, D.; Varoquaux, N.; Nielsen, J. H.; Gustafsson, O.; Root, B.; May, R.; Elson, P.; Seppänen, J. K.; Jae-Joon Lee; Dale, D.; Sunden, K.; Hannah; McDougall, D.; Straw, A.; Hobson, P.; Lucas, G.; Gohlke, C.; Vincent, A. F.; Yu, T. S.; Ma, E.; Silvester, S.; Moad, C. Matplotlib/Matplotlib: REL: V3.7.2, 2023. https://doi.org/10.5281/ZENODO.8118151.

(15) Mcculw. https://github.com/mccdaq/mcculw.

CHAPTER III

Software for Analysis

Introduction

This chapter introduces the concept of data structure within an ion mobility mass spectrometry regime and the unmet needs within this space. This is followed by a description of past work from which this work can continue development from.

Data Structure

In general, a mass spectrometry (MS) experiment will have mass to charge information acquired over a number of scans while an intensity that is proportional to the number of ions that hit the detector is recorded along these two axes. When ion mobility (IM-MS) experiments are completed, an additional axis is added in the form of an arrival-time axis. Arrival times are specific to the mobility chamber in a mass spectrometer, and do not represent the total time that ions are being transmitted throughout the entirty of the mass spectrometer. This arrival-time axis is added to the MS experiment data. With the use of the ptESI source, a solution temperature axis is added alongside the scan time axis, or retention time axis, since they are both recorded as function of time. With collisional-induced unfolded (CIU) voltages and the calculated laboratory-frame energy (LFE), another axis may be added to the experiment if multiple acquisitions are taken or the voltage drop is changed throughout the experiment.

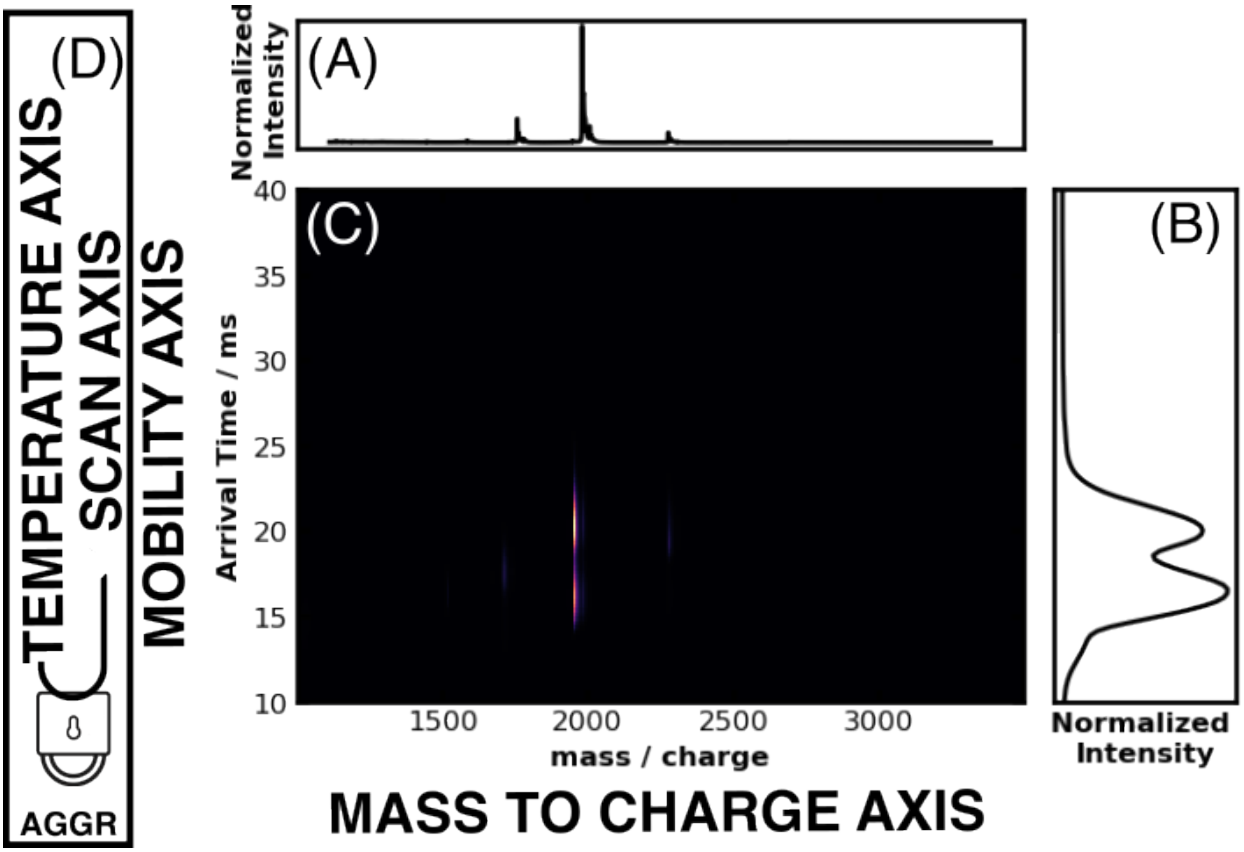


Figure 3.1. There is a mass to charge axis, panel A, and a mobility axis, panel B. This figure shows intensities that have been aggregated across the temperature and scan axes, panel D. Panel C depicts the mobility axis as it relates to the mass to charge axis where the intensity shown has been aggregated across the temperature and scan axes. As shown in panel D, the scan and temperature axes are linked since they both operate as a function of time in minutes.

The data stratus that is accessible using this system, shown in Figure 3.1, contains a wealth of information regarding protein structure. This data shows bins 20 to approximately 100 and mass to charge ratios (*m/z*) ranging from 1000 Da to 3500 *m/z*. The protein depicted is Ribonuclease A (RbA). The trapping voltage of this experiment is 5.0 V which allows for the most native like structure while balancing ion transmission, other data has a myriad of different voltages resulting in various laboratory frame energies. The temperature points range

continuously from 10 °C to 90 °C. This run has 5156 scans and lasted for around 85 minutes. Aggregation can be performed along any axis: scan and temperature, mass to charge, mobility, and laboratory-frame energy. The management of this amount of data can prove challenging.

Unmet Needs

As such, the management, alignment, processing, and visualization of the data is challenging to store over the lifetime need of the investigator to access said data. Additionally, retention of the raw, processed, and visualized data can pose a storage problem. Mass spectrometry unprocessed data, while compressed, still takes up many gigabytes (GBs) of storage space. The ptESI data, which can often be an hour or longer due to probing a temperature range up to 80 °C wide, often occupies 20 or more GBs of storage.

At the start of this project every researcher would have individual Python environments and copies of the jupyter notebooks that were customized. These notebooks were saved at the end of the run for every modification within the notebook for each visualization. This compounds the data storage issue in GBs of storage, but also in tracking changes and modifications. At the beginning, there was a huge demand for version control.

Prior Work on the Pipeline

Previously the processing was broken out into individual folders that contained a jupyter notebook and required the waters utility folder, the ptESI utility folder, the charge states csv, the temperature data file, and the mass spectrometer data file to be moved into the folder for processing. This results in multiple copies of the utility folders, raw data, and even jupyter notebooks to be generated. This provides the opportunity for incorrect versions of the utility folders and outdated functions contained within the jupyter notebook to be used. This initial

system is shown in Figure 3.2. Also regenerating the processing done for individual files proved to be challenging as the version of the utility folders and jupyter notebooks was not recorded in a specific way, requiring the storage of all of the items in a folder to preserve the versions of folders and files used.

As a note, previously temperature alignment to mass spectrometer files was accomplished using interpolation.

```
ptESI-Analysis/
|-- Characterization_Analysis/
    |-- Characterization_workup.ipynb
    |-- README.md
|-- MSAvgChg_VsTemp/
    |-- ProcessedDataFrames/
    |-- ptESI_utils/
        |-- ICReadFunc.py
    |-- [CCYY-MM-DD]_tempdata_[file_tag].txt
    |-- Mass Spectrometry files/
    |-- MSAvgChgVsTemp_Cycling.ipynb
    |-- README.md
    |-- charge_mass_ranges_[protein].csv
    |-- watersutils/
        |-- files from Waters SDK
|-- MSGather_Total/
    |-- ProcessedDataFrames/
    |-- ptESI_utils/
        |-- ICReadFunc.py
    |--[CCYY-MM-DD]_tempdata_[file_tag].txt
    |-- MSGather.ipynb
    |-- Mass Spectrometry files/
    |-- README.md
    |-- charge_mass_ranges_[protein].csv
    |--watersutils/
        |-- files from Waters SDK
|-- MSGraph_Raw/
    |-- Plots/
    |-- ProcessedDataFrames/
    |-- ptESI_utils/
        |-- ICReadFunc.py
    |-- [CCYY-MM-DD]_tempdata_[file_tag].txt
    |-- Mass Spectrometry files/
    |-- MSGraph_Raw.ipynb
    |-- README.md
    |-- charge_mass_ranges_[protein].csv
    |-- watersutils/
        |-- files from Waters SDK
|-- MSSlice_ByScan/
    |-- Plots/
    |-- ProcessedDataFrames/
    |-- ptESI_utils/
        |-- ICReadFunc.py
    |-- [CCYY-MM-DD]_tempdata_[file_tag].txt
    |-- Mass Spectrometry files/
    |-- MSSlice_byScan.ipynb
    |-- README.md
    |-- charge_mass_ranges_[protein].csv
    |-- watersutils/
        |-- files from Waters SDK
|-- MSSlice_ByTemp/
    |-- Plots/
    |-- ProcessedDataFrames/
    |-- ptESI_utils/
        |-- ICReadFunc.py
    |-- [CCYY-MM-DD]_tempdata_[file_tag].txt
    |-- Mass Spectrometry files/
    |-- MSSlice_byTemp.ipynb
    |-- README.md
    |-- charge_mass_ranges_[protein].csv
    |-- watersutils/
        |-- files from Waters SDK
```

Figure 3.2. Original repository layout

Methods

For pipeline development, there is no restriction on the Python version, however there is a version-controlled environment that goes along with the module, as the primary purpose of this custom data analysis module is visualization. Mostly the data is being manipulated in an obstructed fashion, meaning that the data is being manipulated without being seen. As such the Python library pandas is being phased out in favor of the Python library numpy for manipulation. Pandas is still being used to export processed data in a compressed file format. Also of note, is the addition of the Python library ElementTree to handle XML type data, which is now necessary with the advent of the new Graphical User Interface (GUI) of the ptESI-Control library.

Python Libraries

Python Standard Libraries

This pt-Analysis module is built on Python version 3.11.8 and requires the use of the following Python built-in libraries: os, numpy, pandas, ElementTree, datetime, and math.[1] The os library assists with the management of files. Numpy and pandas both handle the importation of files and parse the data contained within the files and further the manipulation of that data. ElementTree allows for the parsing of XML data in particular. Datetime allows for the handling of date-time information. The math library extends the ability of Python to handle more complex mathematical tasks.

Python Imported Libraries

This module requires the use of the following imported libraries: scipy.optimize.curve_fit, scipy.interpolat.interp1d, scipy.signal, and matplotlib.[2,3] It is likely that other scipy functions may be imported in the future as the analytical power of the pt-Analysis

module expands. The scipy modules listed here assist with the handling and manipulation of temperature related data, especially as it relates to mass spectrometry data. Matplotlib is a library that allows for the visualization of data.

Custom Libraries

The Mass Lynx software development kit v4.6 (Waters Co., Wilmslow, UK) is used to extract mass spectral data from mass acquisition files, this will be referred to as watersutils throughout this work. While presently, pt-Analysis is specific to Waters mass spectral acquisition files, there are plans to onboard both the open source mzML file type and ThermoFisher mass acquisition file. There is a custom developed Python module called ptESI_data_utils that contains object-oriented code that will be discussed in the results section.

Other Required Files

Mass spectrometry files are required and should be stored in the data folder in the raw folder. Files stored in the raw folder will retain their state and not be modified throughout running the pt-Analysis module. Associated temperature files should be stored in the raw folder as well. If using the alpha or beta releases of the ptESI-Control module for acquisition, know that there is a method to convert the txt file to an XML file for backwards compatibility support.

Prior to using the pt-Analysis module, evaluate the collected mass spectral data and designate mass to charge regions of interest. For proteins, this may look like identifying the mass ranges for the charge states without the additional adducts, retaining only the primary peak for analysis. There is a template for this file, currently a csv, called "protein_charge_state_ranges.csv", see an example of the contents in Table SX.. If working with

multiple files, like often happens with collisional-induced folding (CIU) workflows, a helper txt file can be made to identify the file names to be gathered together for analysis.

Results and Discussion

This section includes the resulting repository design and code refactor. There is discussion surrounding modification made to the repository and the pre-existing code base. The code refactor section discusses the migration of code that was retained in separate jupyter notebooks into one version-controlled objected oriented programming style utility python file. This file can be accessed via a python script for each manuscript or figure in a manuscript or in a more day-to-day use jupyter notebook that is more customizable as a workspace. The code refactor subsection frames this discussion in steps taken to address preprocessing, general processing, and specialized processing.

Repository Design

The first update to the pt-Analysis codebase was a repository reorganization. The original layout is shown in Figure 3.2. In this repository format, the acquisition files, mass and temperature, were moved into individual folders containing jupyter notebooks for processing alongside the in house watersutils, the in house ptESI_data_utils, and the protein_charge_state_ranges.csv, or the files were copied and pasted. This format required independent file processing, meaning if you needed to do two of the processing tasks contained in two jupyter notebooks, the processing of the mass spectral data needed to happen twice. This is often the longest part of processing due to the challenges listed in the introduction regarding the amount of data contained within. As such, moving to do this extraction a single time for both processing tasks would speed up processing.

With this in mind, a data folder, with raw and processed folders contained within, was introduced to the repository, and all of the jupyter notebooks were migrated into a notebooks folder to allow for all notebooks to access the data notebook in a similar fashion. This decision will support later automation.

Other Modules

Independent module folders were also added to the home directory. This includes the watersutils folder and the ptESI_data_utils folder. Along the way in learning about processing the mass spectral data associated with ptESI-Control type data, another module called waters2numpy was developed in house to support limitations associated with RAM that may be found on common workstations. One of the goals of this project is accessibility, which means that the processing needs to support systems that have less RAM. I completed processing on DDR3 32 GB of RAM. The addition of the waters2numpy processing module empowered reading in over 5000 scans for pt-Analysis. The version control of the ptESI_data_utils folder is associated with the pt-Analysis module and will be discussed in depth here. The version control for both watersutils and waters2numpy is maintained separately and will need to be retrieved from Waters Co., it is free with proof of ownership or warranty of a Waters mass spectrometer, or from the Bush Lab if not found on GitHub, respectively. The waters2numpy module does require use of the Mass Lynx software development kit v4.6 (Waters Co., Wilmslow, UK). Other manufacturers may require different types of preprocessing.

Style File

A reports folder was added to the home directory, and contained within is a figures folder and a style file that contains the parameters used to generate the figures within the folder. The style file contains parameters that are used in house, but can be modified to suit the preferences

of the user. This information was migrated out of the jupyter notebooks for version control, but also to reduce the amount of cluttering type information contained in the juyter notebooks to make it easier to see what is going on in the processing steps. This idea is known as abstraction. The focus of this module is on data processing, and while visualizations are a part of that process, often times a standard format that is suited for academic publishing is sufficient for the in house use cases of this pt-Analysis module. As such removing the parameters from the workspace and instead importing a version-controlled document contain said parameters is sufficient.

```
ptESI-Analysis/
|-- data/
    |-- raw/
        |-- Mass Spectrometry files/
        |-- [CCYY-MM-DD]_tempdata_[file_tag].xml
    |-- interim/
    |-- processed/
        |-- [YYMMDD]_[file_tag]_ProcessedData.h5
        |-- [YYMMDD]_[file_tag]_ptMassAlignment.xml
|-- LICENSE
|-- notebooks/
    |-- MSRaw_Data_Processing.ipynb
    |-- MSAvgChgVsTemp_Cycling.ipynb
    |-- MSGather.ipynb
    |-- MSGraph_Raw.ipynb
    |-- MSSlice_byScan.ipynb
    |-- MSSlice_byTemp.ipynb
    |-- CIU.ipynb
    |-- ptESI-Trial-Comparison.ipynb
    |-- cIMS_Analysis.ipynb
|-- ptESI_data_utils/
    |-- ICReadFunc.py
    |-- ptESI_data_utils.py
|-- references/
    |-- charge_mass_ranges_[protein].csv
    |-- temperature_helper.txt
    |-- voltage_helper.txt
    |-- time_helper.txt
|-- reports/
    |-- figures/
    |-- figures.style
|-- waters2numpy/
    |-- src/
        |-- waters2numpy.py
    |-- README.md
|-- watersutils/
    |-- files from Waters SDK
|-- [experiment].py
|-- environment.yml
|-- README.md
|-- requirements.txt
```

Figure 3.3. New repository layout

These changes resulted in a repository that contains a single copy of each document that is version-controlled either by this pt-Analysis module or by another version-controlled module, as can be seen in Figure 3.3. Where the color coding matches that of Figure 3.2. This approach also has the benefit of assisting with long-term data retention. Since the pt-Analysis module is version controlled, when a publication, like this one, is made the version of the module can be listed alongside the publication and should figures or additional data need to be interpreted, then

that same version of the codebase can be used. This type of structure will allow for historical context to be applied to future work. Additionally, it means that only the code version and the raw data need to be stored, and that figures can become optional storage.

Code Refactor

After reorganizing the repository, all of the code contained in the various jupyter notebooks needed to be organized into a format that allows for automation and version control. As such, raw code was migrated into functions. Those functions were then migrated into classes. These classes were migrated into the ptESI_data_utils.py file contained in the ptESI_data_utils folder.

The ICRead_Func.py file was used at the beginning of the project for some notebooks and is included to retain backwards compatibility. The only function inside handles extracted charge states from the total ion chromatogram and storing this extracted ion chromatogram type data in an interim or processed data file depending on the notebook accessing the extracted ion chromatogram information.

Currently, at the time of writing, with the new advent of the waters2numpy module still in troubleshooting and this historic py file, the processed mass spectrometry data storage method is in flux, with some uses in the historic and some in the new waters2numpy, namely the mobility type data for ptESI-Control experiments. The processed file type will be h5. This report will focus primarily on processing the temperature associated data with the understanding that the processed data can be read into the workspace alongside either the historic py file mass processing or the waters2numpy file mass processing.

After creating the various functions needed to recreate the workflows inside the jupyter notebooks that I was more familiar with, including: MSAvgChgVsTemp_Cycling, MSGather, MSGraph_Raw, MSSlice_byScan, and Characterization_workup, I noticed some common themes. Mass data needed to be extracted using watersutils and temperature data, where present, needed to be aligned with the extracted mass data. After this initial mass data extraction, some processing into an extracted ion chromatogram from a total ion chromatogram may take place, almost always with extracting charge states using the protein_charge_state_ranges.csv file. The MSGraph_Raw visualized the total ion chromatogram. The MSGather and MSSlice_byScan notebooks both visualize extracted ion chromatograms, via charge states and via user defined scan intervals respectively. The MSAvgChgVsTemp_Cycling notebook also utilizes the extracted ion chromatogram for charge states.

This led to the development of the various classes. There is further delineation based on the type of data being processed as well. Temperature data alone and mass data alone have their own class, ptlog and masslog respectively and the combination of mass and temperature data is ptmasslog. There is a file processing class and a data processing class.

The file processing class reads in the various files in the repository so that files can be read into the workspace and exported from the workspace, such as extracting the mass data from the raw data folder and then getting the extracted ion chromatograms of the charge states and exporting these extracted ion chromatograms to an h5 file to the processed data folder and sending the MSGraph_Raw visualizations to the figures folder. The file processing class handles file manipulation including the repository folders and also the names of the files in the raw data folder. These files in the raw data folder are differentiated using an algorithm that handles all of the files present and sorts them into mass data, temperature data, and other types of data.

Currently this is based on the Waters proprietary naming conventions and can be updated to the system in use.

Preprocessing

The data preprocessing class has many subclasses including the aforementioned ptlog, masslog, and ptmasslog. In the masslog class is the WatersSDKInternal class which actually houses most of the mass extraction functions. The reasoning behind this choice is that other manufacturers can be added as additional subclasses to the masslog class, including the open source mzML file processing. As a subclass to the WatersSDKInternal class is an instrument specific class, cyclicInternal class. Sometimes manufacturers will update their file organization, so for the file organization structures that are specific to a generation of instrument, subclasses can allow for support of all generations of mass spectrometers in a laboratory. In our case, this was implemented to support onboarding the Waters Synapt G2 HDMS hybrid mass spectrometer (Waters Co., Wilmslow, UK) modified with an RF-confining drift cell containing nitrogen gas, as described previously[4] for analysis utilizing this analysis system.

Additionally, there is a mobilitylog subclass to the data preprocessing class, as it has become more common to collect ion mobility data alongside the mass spectral data. The masslog classes focus primarily on data that only intensity for mass channels. The mobilitylog class needs to be broken out from the masslog class due to a change in how the data is structured. Along with intensity for mass channels, an additional metric, mobility bins, which are measured in milliseconds, needs to be considered. Generally, mass channels are aggregated and intensity as it relates to arrival time is displayed, as seen in Figure SB1. However the ability to use the masslog class to showcase intensity as it relates to mass to charge ratio is retained. In this case, the mobility bins are aggregated, displaying intensity as it relates to mass to charge ration.

The ptmzlog is another subclass of the data preprocessing class that allows for the pt-Control output data to be used in tandem with mass spectral data. The sampling rate of the mass spectrometer is likely different than the sampling rate of the temperature controller used in the ptESI-Control experiment. Due to a differing sampling rate, an alignment of data is required. Additionally, it is likely that the temperature controller and mass spectrometer have different start times due to the manual nature of having to hit a start button for both the temperature controller software and the mass spectrometer software. There is future potential for automation utilizing the automation software for the mass spectrometer, especially in a liquid chromatography injection regime, we currently utilize direct injection, and further development on the ptESI-Control GUI to enable automation. Due to both the manual start of both systems and the differing sampling rates, an alignment of the temperature data to the mass spectral data is necessary.

Alignment of Temperature and Mass Acquisition Data

A visual of the reason for alignment is present in Figure 3.4. Along the mass axis, there are a number of events that record mass to charge at a specific time point, retention time in minutes. Along the temperature axis there are a different number of events that record temperature at a specific time point. The incident time points for both mass and temperature will likely differ by a number of seconds, which is important when one of the recorded metrics, namely ion mobility, has a value that is several magnitudes smaller than the minutes that retention time per mass to charge scan is recorded.

The incident time alignment was completed prior to this work. The previous temperature alignment was completed using interpolation, which may accurately give the temperature at that time point, but will not necessarily reflect the temperatures that the ions accumulating for the

next mass to charge recording event were subjected to. The aim of the temperature alignment in Figure 3.4 aims to address this. This technique aims to implement a weighted sum approach to accurately reflecting the temperatures that the ions were subjected to in the accumulation phase for the mass to charge recording event. As such, the first identified time cutoff reference is the end of the mass to charge recording event. Meaning that temperature after the mass to charge recording event are not considered for that recording event. For example, in Figure 3.4, the Y time duration in the mass axis is reflected in the alignment of temperature by only considering time points in the temperature axis prior to the F time duration. Additionally temperature duration considered for a different time duration on the mass axis, such as X in Figure 3.4 along the mass axis, will not be considered for the temperature alignment for mass duration Y. Meaning that the entirety of A, the entirety of B, and the first portion of C will not be considered in the temperature alignment for the mass duration Y. The temperature alignment for mass duration Y will consider the second portion of temperature for temperature duration C, the entirety of temperature duration D, and the first portion of temperature duration E.

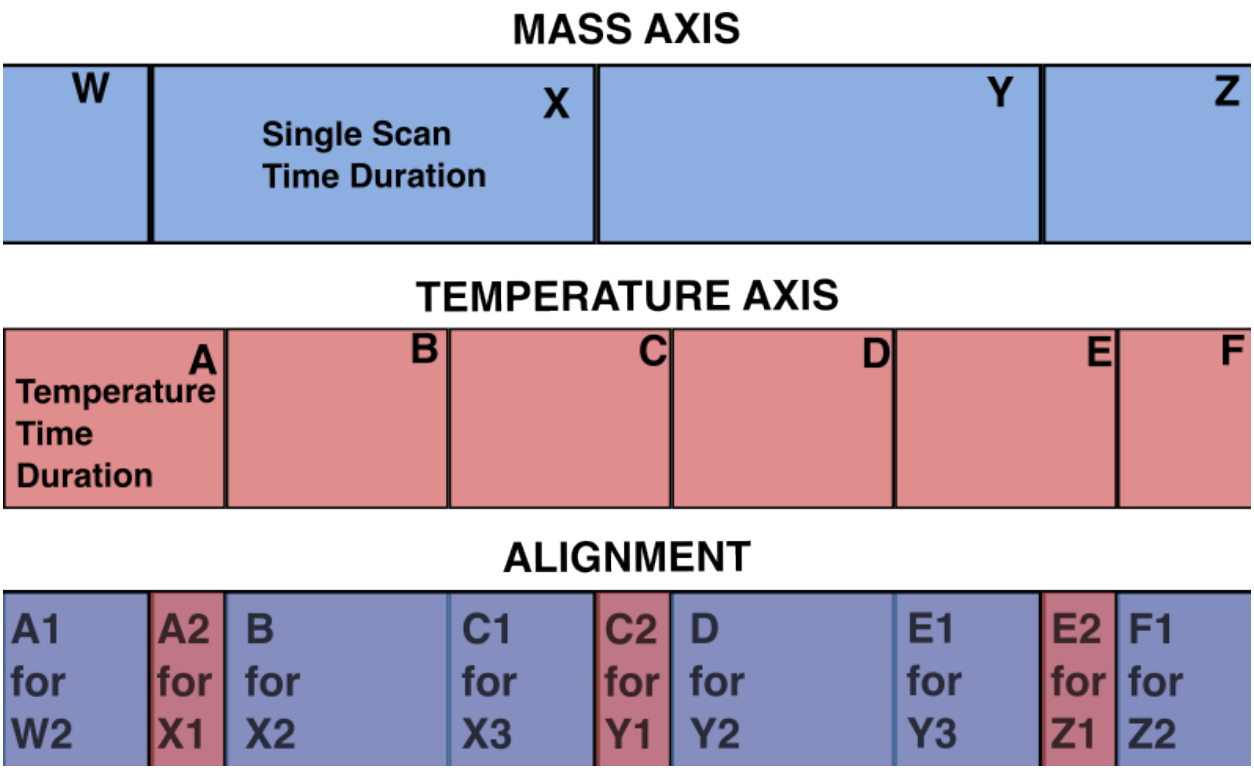


Figure 3.4. There is a mass axis where single scans have a duration and a temperature axis where temperature is steady for a duration. The sum happens along the mass axis, meaning that labels W, X, Y, and Z have the sum aggregation of solution temperature data. The multiplier is provided by the temperature axis with label A, B, C, D, E, and F. The duration creates the weights, both scan time and temperature time duration are used flexibly throughout the total run time in order to generate the weight. Once the weight is known it is multiplied by the multiplier and these values are summed along the W, X, Y, and Z mass axis. This sum is exported alongside the scan number.

Due to the mass to charge recording event timing often bisecting the temperature recording event timing, a temperature point may be applied twice for two different scans as in the first portion of C in Figure 3.4 along the temperature axis being applied to mass to charge recording event X and the second portion of C being applied to the next mass to charge recording

event Y as well. However, the bisection is almost never equal and may vary in duration. So the duration of the temperature axis C is split unevenly between mass scan X and Y. As such, the applied temperature duration is used as a weight multiplied by the temperature. Each fractional temperature for a mass time duration is summed, creating this weighted sum approach.

To implement this weighted sum in a flexible fashion, a customized algorithm was developed to also handle edge cases such as the first mass scan duration and the last mass scan duration, and instances where there are a varying number of temperature time durations for each mass scan duration. This phenomenon is likely caused by latency in data transmission. In Figure 3.4, the mass time duration and the temperature time duration are shown as being equal throughout, but in practice this is often not the case, which creates space and time for an additional temperature duration needing to be accessed within a single mass time duration. This weighted sum is a method in the ptmzlog class, allowing for other methods to be developed in the future such as the implementation of a Riemann sum approach.

After this alignment is completed, a temperature for each scan is exported in an XML file that can be read into the workspace to create a one-to-onealignment between either the scan number or the retention time for each scan in the masslog.

All of these tasks are described as preprocessing in the codebase and are stored and accessible in the data preprocessing object. There are *m*/*z* data related tasks and temperature data related tasks, and the combination of the two in the preprocessing class, as described in Figure 3.5. There are two other primary classes including the general processing class and the specialized processing class, also depicted in Figure 3.5.

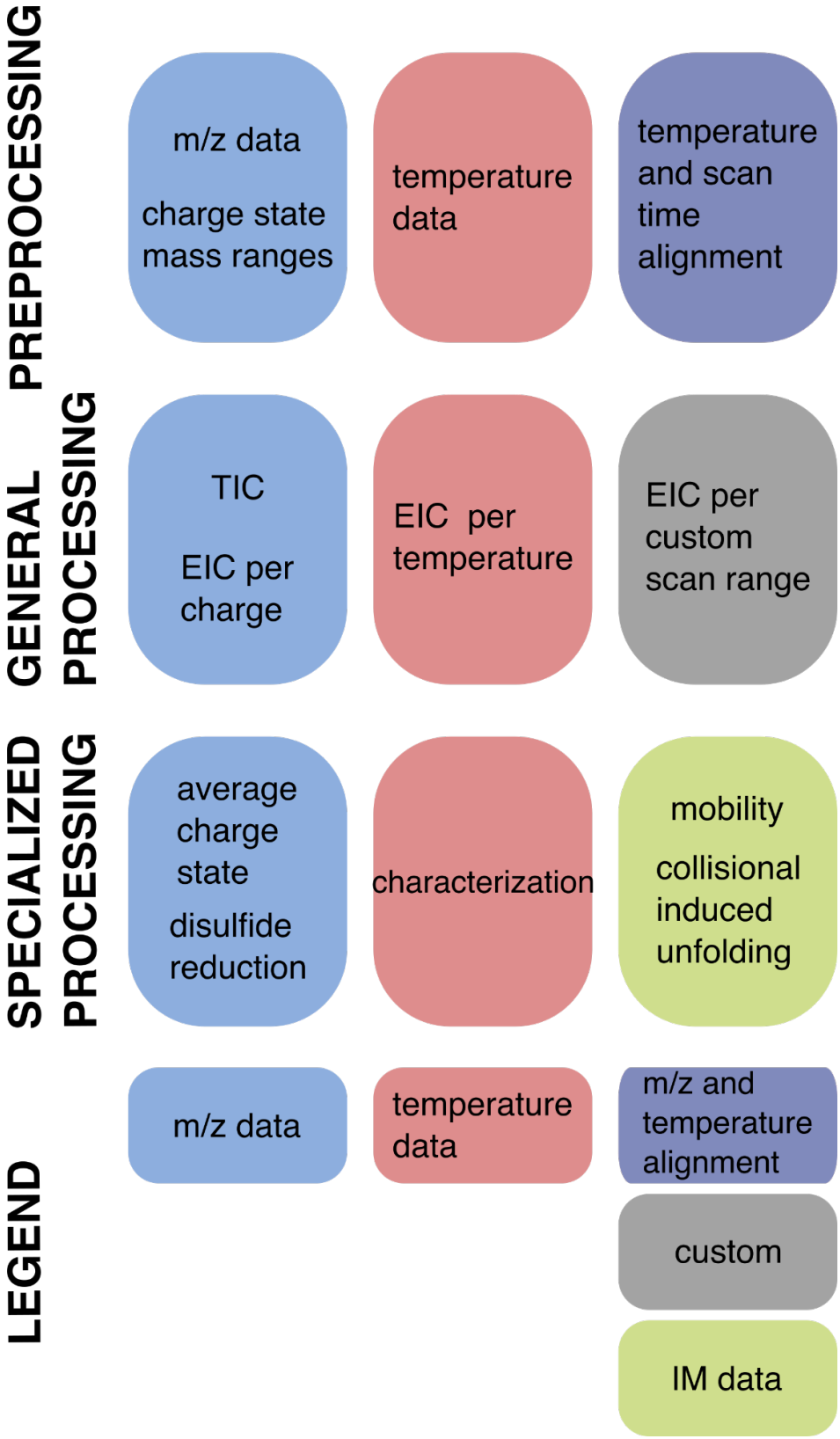


Figure 3.5. pt-Analysis processing module components.

General Processing

In the generalized processing class, there a many functions related to generating visualizations for a total ion chromatogram and for generating visualizations that extract information from the total ion chromatogram, see an example in Figure SB1, either extracting charge state information, see an example in Figure SB3, or the mass-spectral information for a specific temperature, see Figure SB2 for an example, or for a customized range, often used to check the status of the mass spectral data throughout the run.

Records of Run

This collection of methods in the general processing class can effectively recreate the real-time readout displayed in the acquisition software of the mass spectrometer. This means that the general processing class can years down the road display what happened throughout the experiment, either to remind the user what happened or to show to others their observations in real time. This class has been used to effectively show an entire mass spectral acquisition using a gif or mp4 file format. This is especially effective in conjunction with ptESI-Control experiments to watch the mass spectral data change as time progresses, allowing for the observation of additional charge states as temperature shifts. An example of this type of generalized processing is in Figure SB1.

The Characterization_workup visualizations were migrated into the ptlog class as they only utilize temperature information. There are two visualizations of note, the first plots the temperature program against the thermistor readback and the heat sink temperature along with the power and current draw, which can be seen in Figure SB5. This visual provides information about experimental temperature and theoretical temperature. It also shows some information about the power draw, which can help with troubleshooting if there seems to be thermal lag. The

current draw shows the function of the solid-state heat pump, again empowering troubleshooting of the hardware of the ptESI-Control temperature controller. The second visualization specifically investigates the difference between the experimental temperature in the thermistor readback and the theoretical temperature in programmed temperature, and an example of this visual can be seen in Figure SB6.

Specialized Processing

Specialized processing often requires specific types of information. This type of processing requires diving deeper into the metadata to extract experimental run information, especially in the case of ion mobility. While creating the arrival time distribution for the entire run, where all scans are aggregated together, is not difficult, there have been some interesting challenges in accurately representing the arrival time axis.

One of the challenges involves reliably extracting the separate step information. Ion mobility often has an injection time, a separate time, and an ejection time. Since mobility is a time bound dimension, ensuring that these times, injection, separate, and ejection, are reported accurately is crucial in accurately reporting the arrival times which can lead to a collision-cross section calculation. Ensuring that this size is appropriately represented is important to understanding protein stability, especially if that collision-cross section shifts alongside the temperature in a ptESI experiment.

The other challenge posed by the ion mobility processing relates the discussion in the unmet needs section surrounding the amount of data that needs to be processed. It has been rare that my RAM has not been maximized when processing multiple data files. Little by little improvements to the data analysis pipeline have reduced the amount of time that I have spent waiting for RAM processing. One of these improvements was reorganizing the repository to

support more complex analysis, but many of the improvements have been optimizing prior code throughout the refactor and update process.

To the average charge state associated specialized processing, the ICRead_Func.py functions preform the charge state extraction, the inputs to the function were updated to align with the new repository structure. This function contains code to generate both the extracted ion chromatogram processed mass spectral data file and for a visualization of the selected mass spectral information that is selected for each charge states to migrate to the extracted ion chromatograms, as seen in Figure SB3. This useful quality check type visual was moved into its own function so that it may be accessed separately from creating the extracted ion chromatograms. I developed some additional visualizations surrounding the tracking of charge state abundance over time.

Conclusions

This specialized processing is highly collaborative and ever changing as experiments and their results guide the need for certain types of post processing and visualizations. Ultimately the class structure allows for additional expansion to support the ever-changing needs of the user. It would be ideal to at the beginning of an experiment to create an executable Python file or a series of Python files that are directly associated with the needs of the experiment. These Python files can be stored alongside the raw data files for long term data storage. To develop these Python files, a jupyter notebook can be utilized to explore the available figures. Once the jupyter notebook on a fresh, just opened kernel, can run every cell without error, each code containing line can be put into a new Python file which can be run in an integrated development environment, such as Visual Studio Code, which is used in house, or can be run directly from the

command line. As for future work, some effort for processing multiple files with one command has been attempted, but batch processing is necessary.

Acknowledgements

This work was supported by the National Science Foundation through award 2203513 (Bush) from the Division of Chemistry, with partial co-funding from the Division of Molecular and Cellular Biosciences.

References

(1) Python. https://docs.python.org/release/3.11.8/.
(2) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; Van Der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; Van Mulbregt, P.; SciPy 1.0 Contributors; Vijaykumar, A.; Bardelli, A. P.; Rothberg, A.; Hilboll, A.; Kloeckner, A.; Scopatz, A.; Lee, A.; Rokem, A.; Woods, C. N.; Fulton, C.; Masson, C.; Häggström, C.; Fitzgerald, C.; Nicholson, D. A.; Hagen, D. R.; Pasechnik, D. V.; Olivetti, E.; Martin, E.; Wieser, E.; Silva, F.; Lenders, F.; Wilhelm, F.; Young, G.; Price, G. A.; Ingold, G.-L.; Allen, G. E.; Lee, G. R.; Audren, H.; Probst, I.; Dietrich, J. P.; Silterra, J.; Webber, J. T.; Slavič, J.; Nothman, J.; Buchner, J.; Kulick, J.; Schönberger, J. L.; De Miranda Cardoso, J. V.; Reimer, J.; Harrington, J.; Rodríguez, J. L. C.; Nunez-Iglesias, J.; Kuczynski, J.; Tritz, K.; Thoma, M.; Newville, M.; Kümmerer, M.; Bolingbroke, M.; Tartre, M.; Pak, M.; Smith, N. J.; Nowaczyk, N.; Shebanov, N.; Pavlyk, O.; Brodtkorb, P. A.; Lee,

P.; McGibbon, R. T.; Feldbauer, R.; Lewis, S.; Tygier, S.; Sievert, S.; Vigna, S.; Peterson, S.; More, S.; Pudlik, T.; Oshima, T.; Pingel, T. J.; Robitaille, T. P.; Spura, T.; Jones, T. R.; Cera, T.; Leslie, T.; Zito, T.; Krauss, T.; Upadhyay, U.; Halchenko, Y. O.; Vázquez-Baeza, Y. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17* (3), 261–272. https://doi.org/10.1038/s41592-019-0686-2.

(3) Team, T. M. D. Matplotlib: Visualization with Python, 2023. https://doi.org/10.5281/ZENODO.8347255.

(4) Zercher, B. P.; Gozzo, T. A.; Wageman, A.; Bush, M. F. Enhancing the Depth of Analyses with Next-Generation Ion Mobility Experiments. *Annu. Rev. Anal. Chem.* **2023**, *16* (1), 27–48. https://doi.org/10.1146/annurev-anchem-091522-031329.

CHAPTER IV

Combining ptESI and Collision-Induced Unfolding

Introduction

Native mass spectrometry (MS) has a helpful tool in the bioanalytical space, especially regarding structure of macromolecules, including proteins[1]. Ion mobility (IM) is a gas-phase structural tool that is of use in the bioanalytical space, especially in the case of proteins. In IM, charged ions experience collisions with a neutral background gas as they move through a drift region containing an applied electric field. The time that the ion spends in the drift region is inversely proportional to the mobility of the ion and may be used to acquire a rotationally averaged metric collision cross section (CCS), which is a structural parameter that is representative of the surface area of the ion.[2] In short, IM-MS can separate ions based on their shape, size, and charge. Results from IM-MS studies have been used to elucidate copopulated protein conformers for insights into protein structures utilizing charge state ions and CCS. Protein CCSs have proven to be challenging to capture due to the inherent dynamicity of this type of biomolecule. This difficulty in uncovering protein CCSs allowed for the development of collision induced unfolding (CIU), which has been documented to observe gas-phase shifts in stability.[3,4] In a CIU measurement ions are activated in the gas phase via a stepwise ramp in collisional activation. As a biomolecular ion is activated, it unfolds, changing the drift time measuring in IM. For the stepwise ramp in collisional activation, the IM-MS measurements will record the ion unfolding as a function of collisional activation as compared to a change in ion drift time or CCS, as can be seen in Figure 4.5. The unfolding pathway in a CIU measurement

represents the stability of the ion, and structural changes of an analyte can result in detectable shifts in this stability. CIU has been shown to reflect solution phase stability shifts in antibodies,[5–7] even those with disulfide bonds.[8]

Temperature-controlled electrospray ionization (tcESI) sources have been utilized to modify the temperature of a solution containing sample in the ESI capillary and when detecting using MS have resulted in probing the stability of noncovalent complexes.[9] Variable-temperature ESI (vtESI) investigating the (un)folding of protein monomers and complexes and the transition state thermochemistry of IgG antibodies have previously been completed. [10,11] Many temperature-controlled sources use resistive heating or thermoelectric devices and experiments are performed in a stepped temperature fashion. Incubation is performed at a selected temperature and spectra are acquired after each incubation step. [12–18] Rapid acquisition using "digital heating" has been completed previously. [19] In these techniques the metric of average charge state is used to characterize the (un)folding of proteins when analyzed using MS. The protein will unfold with increasing temperature allowing for a rise in surface area on the protein. This allows for additional protonation sites to become accessible, thereby shifting the charge state distribution to higher charge states. This data is often visualized in average charge state versus temperature plots similar to Figure 4.2. The sigmoidal response of average charge state with respect to temperature can be fit, described previously,[20] and the inflection point of the curve calculated for to yield the midpoint transition temperature (T_m). Alternatively, folded and unfolded charge state distributions may be used to find the T_m by calculating the relative abundance of each with respect to temperature and finding the point at which the folded and unfolded distributions intersect[19]

Materials and Methods

Sample Preparation

Ribonuclease A (bovine pancreas, R6513) was purchased from Sigma-Aldrich (St. Louis, MO). The RbA stock solution was prepared at 50 µM in aqueous 200 mM ammonium acetate at pH 7 and exchanged (final concentration 20 µM) into fresh aqueous ammonium acetate using Micro Bio-Spin 6 columns (Bio-Rad, Hercules, CA) in order to desalt. The final protein concentration of RbA was 10 µM, achieved by dilution using 200 mM ammonium acetate at pH 7, for solutions subjected to MS analysis.

ptESI and Mass Spectrometry Settings

Five µL of solution was loaded into a borosilicate glass capillary (1.00 mm O.D., 0.78 mm I.D.) that was pulled to a 1-3 µm tip (Sutter Instruments P-97), then inserted into the ptESI source held at 10 °C for ptESI experiments and 25 °C for CIU and ptESI-CIU experiments. The ptESI source and a Waters SELECT SERIES Cyclic IMS system[21] were used for all experiments. A platinum wire electrode was used to make electrical contact with the solution for electrokinetic ESI[22] for all experiments. Electrospray was established by applying 0.4 to 0.8 kV of potential to the electrode. Mass spectra and ion mobility data were acquired at a rate of 1 scan·second^{-1}. Mass spectra were calibrated using ESI-L LCMS Tuning Solution (G1969-85000, Agilent, Santa Clara, CA). Mass spectral data was acquired using the associated flexible instrument control software[21] and extracted for analysis using the Mass Lynx software development kit v4.6 (Waters Co., Wilmslow, UK). Temperature data was acquired using the ptESI-Control platform. Data analysis was completed using the extracted mass spectral data alongside custom Python tools developed in house using the pt-Analysis platform.

For ptESI experiments, a temperature gradient of 10 °C·min^{-1} going from 10 °C to 90 °C with 1 minute of incubation at the following temperatures: 10, 25, 37, 45, 60, 75, and 90 °C, were used to evaluate protein unfolding, shown in Figure 4.1. The extracted ion chromatograms for each charge state are generated using specified mass ranges and the average charge state is calculated using the intensity of the extracted ion chromatogram as a weight for each charge state. Temperature readings from the embedded thermistor in the ptESI source are used to link experimental time and block temperature.

For CIU experiments, at 25 °C, arrival time distributions of RbA were measured using the traveling wave cyclic ion mobility cell filled with 2.24 mBar nitrogen. CIU was monitored in the trap cell filled with Argon as a function of the voltage drop used to accelerate ions into the trap cell prior to IM separation. The monitored voltage drops included: 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 14.5, 15.0, 15.5, 16.0, 16.5, 17.0, 17.5, 18.0, 18.5, 19.0, 19.5, 20.0, 25.0, 30.0, 40.0, 50.0, 60.0, 70.0, and 80.0 V. RbA charge state 7+ was included for this analysis. Since RbA 7+ showcases two features on the arrival time distribution, an early feature and a late feature, with an inflection point, this inflection point can be used as a division between the early and late feature, as showcased in Figure 4.8, Panel A. Arrival times prior to this inflection point in this study were considered to be more folded and arrival times after this inflection point in this study were considered to be more unfolded. Using the normalized intensity, calculated by taking the intensity of the extracted 7+ arrival time feature divided by the intensity of all arrival times, the more unfolded population, or the later arrival time feature, is taken to be a fraction of the total population, which contains both folded, earlier arrival time features, and unfolded, later arrival time features, populations. This fraction in this study was considered to be the relative unfolded

population. This relative unfolded population amount was related to the monitored voltage drop in the trap region of the Waters SELECT SERIES Cyclic IMS system[21].

For ptESI-CIU experiments, the solution temperature was set to 25 °C initially and then moved to one of the following experimental temperatures: 10, 25, 37, 45, 60, 75, or 90 °C prior to the beginning of acquisition. The voltage drop was set to 5.0 V initially and then moved to one of the following experimental voltage drops: 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 14.5, 15.0, 15.5, 16.0, 16.5, 17.0, 17.5, 18.0, 18.5, 19.0, 19.5, 20.0, 25.0, 30.0, 40.0, 50.0, 60.0, 70.0, or 80.0 V. Every combination of experimental temperatures and voltage drops were acquired. The relative unfolded population as it relates to the monitored voltage drop in the trap region of the Waters SELECT SERIES Cyclic IMS system[21] was tracked alongside the solution temperature.

While most of these figures were produced using in house Python software, further edits were made using Inkscape[23] and GIMP[24].

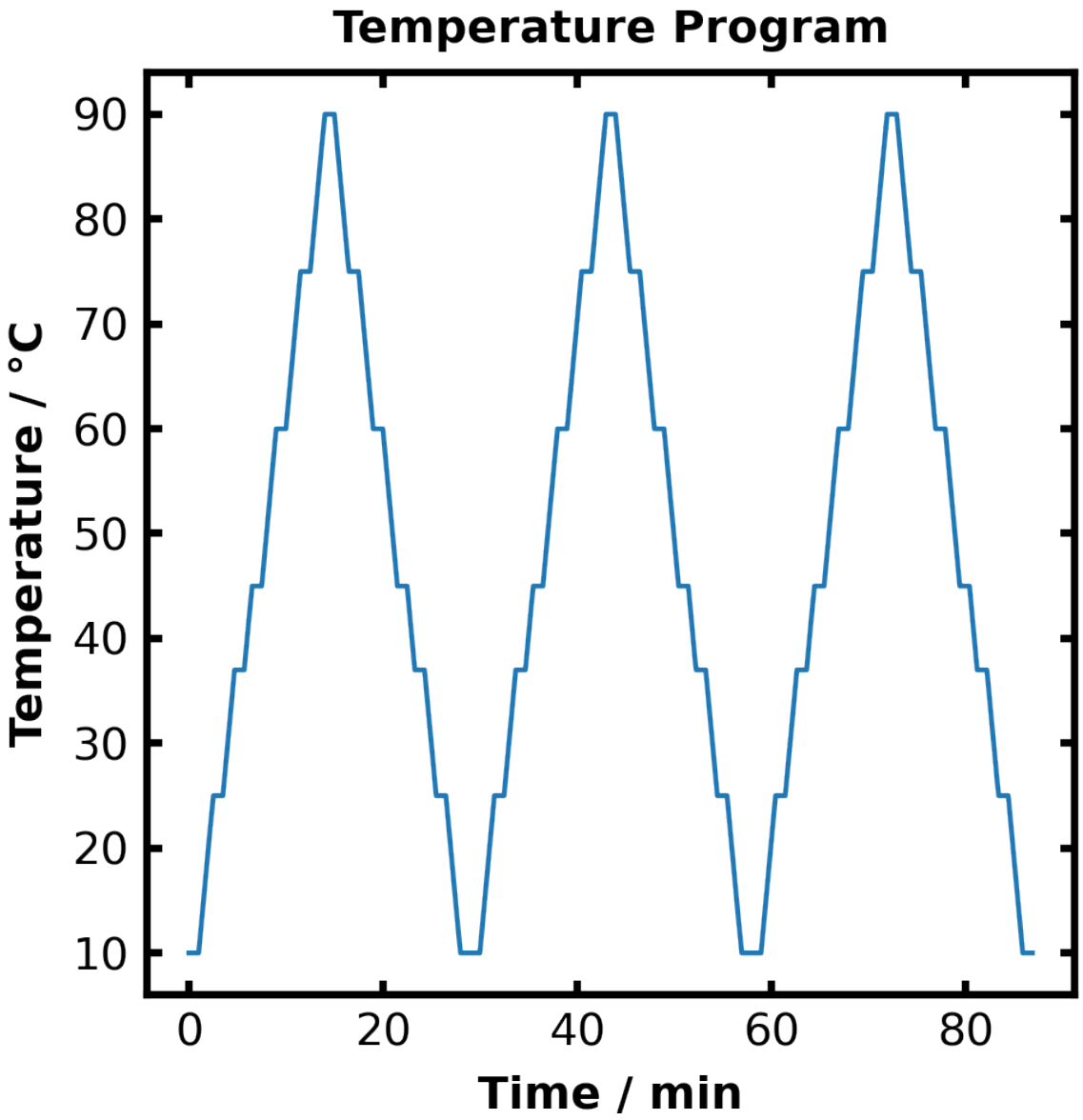


Figure 4.1. Temperature program used in ptESI based experiments

Results and Discussion

Using ptESI to Study Protein Unfolding

In previous work,[20] both lysozyme and ubiquitin exhibited an unfolding curve visualized by comparing average charge state to temperature. The curves were repeatable over a number of

cycles and between replicates with a tight sigmoidal response of average charge state to temperature. This work aimed to add to the body of proteins studied by evaluating Ribonuclease A (RbA).

Initially, with similar testing conditions to past work, an average charge state response as a result of temperature change was recorded. Variation in the sigmoidal distribution as the temperature increased was observed, with less response from the average charge state compared to past work. With RbA, the average charge state ranges only from around 6.8 to 7.8, a spread of a single additional charge from the beginning of the cycle at low temperature to the end of the cycle at higher temperatures, as seen in Figure 4.2. Average charge state is calculated from the extracted ion chromatograms for each charge state where the intensity functions as a weight.

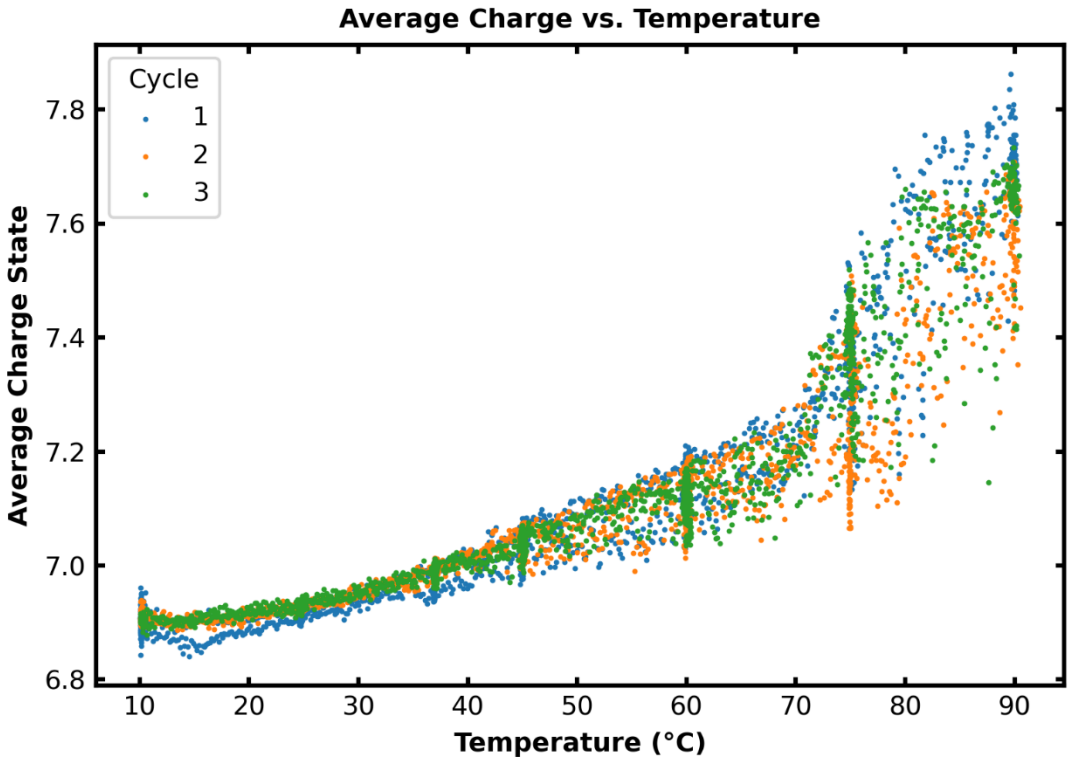


Figure 4.2. Average Charge State vs Temperature plot for RbA at 7.0 pH. The temperature program used in this experiment ranged from 10 °C to 90 °C with 60 second holds at 10, 25, 37, 45, 60, 75, and 90 °C. The additional density seen at these temperatures can be explained by the additional number of data points collected during the hold period.

pH

The lack of an upper asymptote may suggest that RbA is too stable to fully unfold with these experimental conditions. Past work[20] has successfully decreased pH in lysozyme to destabilize the structure in order to fully unfold. This, however, does not explain the variation in average charge state distribution at higher temperatures as compared to lower temperatures. RbA 1 from pancreatic bovine sources and RbA 1 in humans have a different optimal pHs.[25] While the selected pH of 7 is closer to the native pH in humans, pH = 7.3, it is not similar to the

biologically active pH of pancreatic bovine RbA, pH = 6.0.[25] This corroborates the indication that the missing upper asymptote in Figure 4.2 indicates that the experimental pH is too high and should this ptESI experiment be repeated, a pH of 6.0 for bovine pancreatic RbA 1 should be selected or that a pH of 7.3 should be used with a new source of bovine brain RbA for activity most similar to that of human RbA 1. A pH of 7 for pancreatic bovine RbA was selected in part to preform intact analysis to compare to other work[26] completed in RbA reduction. The variation of average charge state as compared to temperature may be explained in part by utilizing a non-optimal pH for enzymatic activity of bovine pancreatic RbA 1. Additionally, there may be some introduced variation in the gas phase pH as a result of the ESI process.[27]

Ion Mobility

After observing the variation in average charge state, to further probe sources of variation, ion mobility data was collected, and the resulting arrival-time distribution (ATD) is shown in Figure 4.3. This additional metric was selected because average charge state as it relates to temperature has been shown to be similar to folded and unfolded abundance as it relates to temperature. IM-MS type data was added because ion mobility has been shown to shift alongside average charge state as they relate to temperature.[16] The arrival time distribution of each charge state through the duration of a ptESI experiment can be seen in Figure 4.3. As can be seen in Figure 4.3, the ATDs for many of the charge states exhibit multiple peaks, corresponding to an early arrival time and a late arrival time, and for charge state 6+, an even later arrival time feature can be seen. This suggests that over the duration of a ptESI experiment that folded, unfolded, and extended structures may be accessed. The 7+ charge state for RbA was selected for additional analysis because it had the highest relative abundance. The ATDs of 7+ RbA for different temperature states throughout a ptESI experiment can be seen in Figure 4.4. For the

early arrival time feature, at lower temperatures the peak is more intense, and as the temperature increases the late arrival time feature increases in intensity. This suggests that the early arrival time feature is folded, and the late arrival time feature is unfolded, as it is known that at higher temperatures there is a higher average charge state suggesting more available surface on which a charge may be applied. While this is the general trend, temperatures 75 °C and 90 °C may be showing a different subpopulation within the folding and unfolding curves. This is evidenced by the shift in midpoint for both of these curves as compared to other temperature states. This deviation in ATD may explain the variance in average charge state at higher temperatures seen in Figure 4.4. This deviation merited additional investigation. Collisional-induced unfolding (CIU) was onboarded to the ptESI system in order to understand the additional subpopulations of RbA that may be present, as has been shown to be effective in prior work.[28]

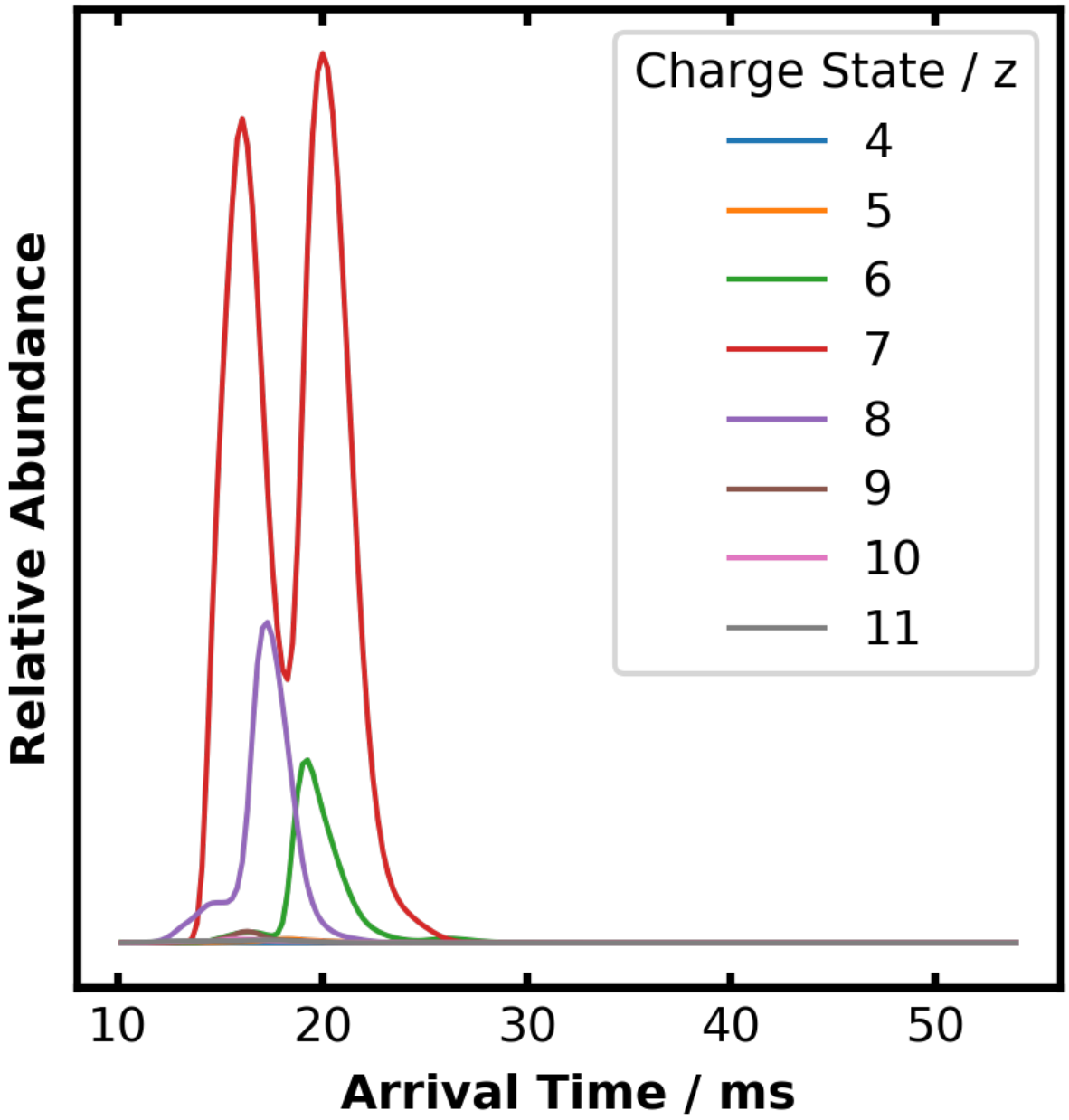


Figure 4.3. Arrival time distribution (ATD) for RbA throughout a cycling experiment. The temperature program used in this experiment ranged from 10 °C to 90 °C with 60 second holds at 10, 25, 37, 45, 60, 75, and 90 °C.

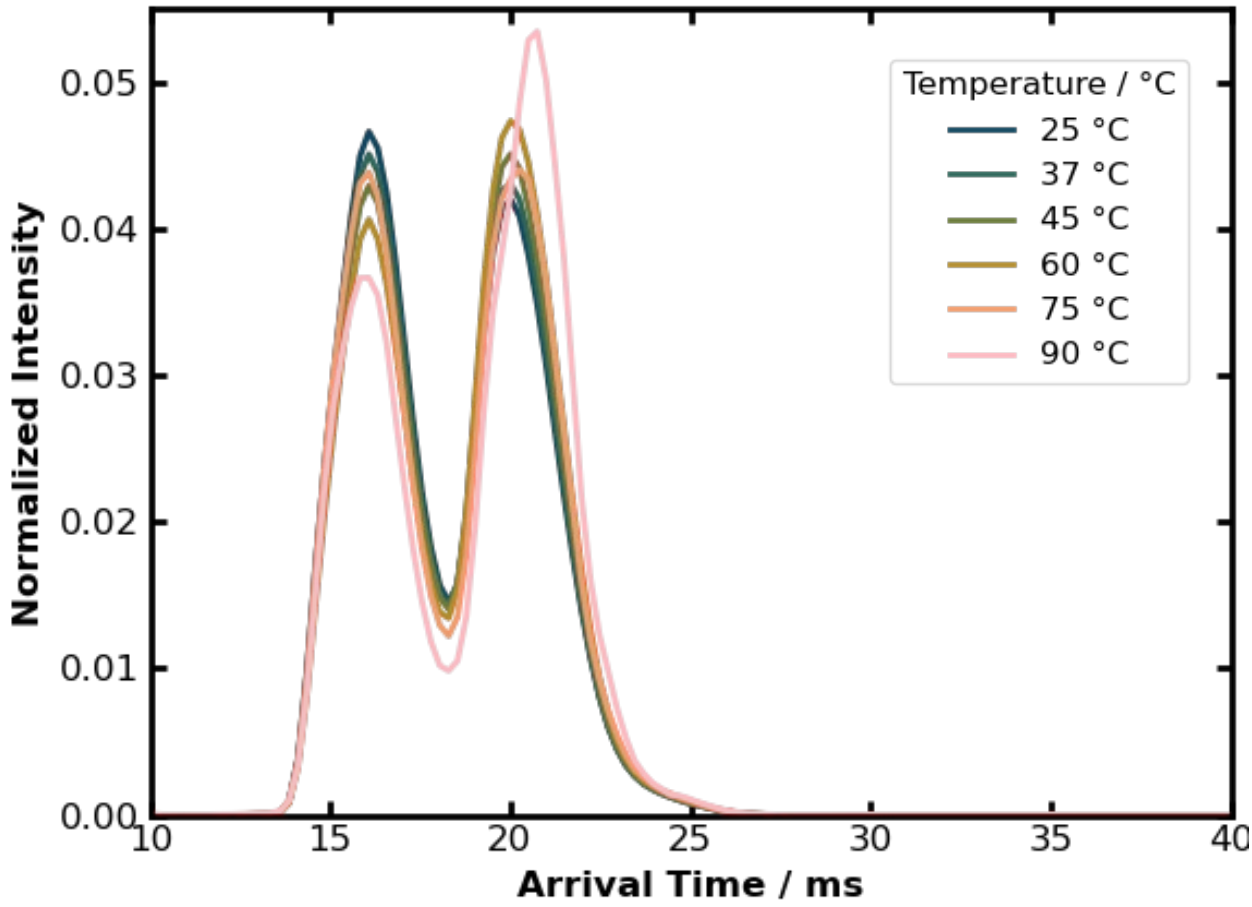


Figure 4.4. ATD for 7+ RbA for during a cycling experiment with a temperature gradient of 10 °C·min^{-1} going from 10 °C to 90 °C and 1 minute of incubation at the following temperatures: 10, 25, 37, 45, 60, 75, and 90 °C for three cycles. The 1-minute incubations were extracted for each temperature condition to construct this ATD.

Using CIU to Study Protein Unfolding

At 25 °C, selected laboratory frame energies (LFE) were evaluated from the total list of applied trap voltage biases as described in the experimental section, results in Figure 4.4. Again, the 7+ charge state was chosen for evaluation. Laboratory frame energy was calculated as the charge state multiplied by the applied trap voltage bias. As with the mobility data for the ptESI experiment, two features can be seen in the ATD, an early arrival time and late arrival time

feature. As LFE increases, the early arrival time feature depletes and the late arrival time feature increases. As with the mobility data observed with ptESI, multiple subpopulations in the late arrival time feature with earlier or later arrival times are present.

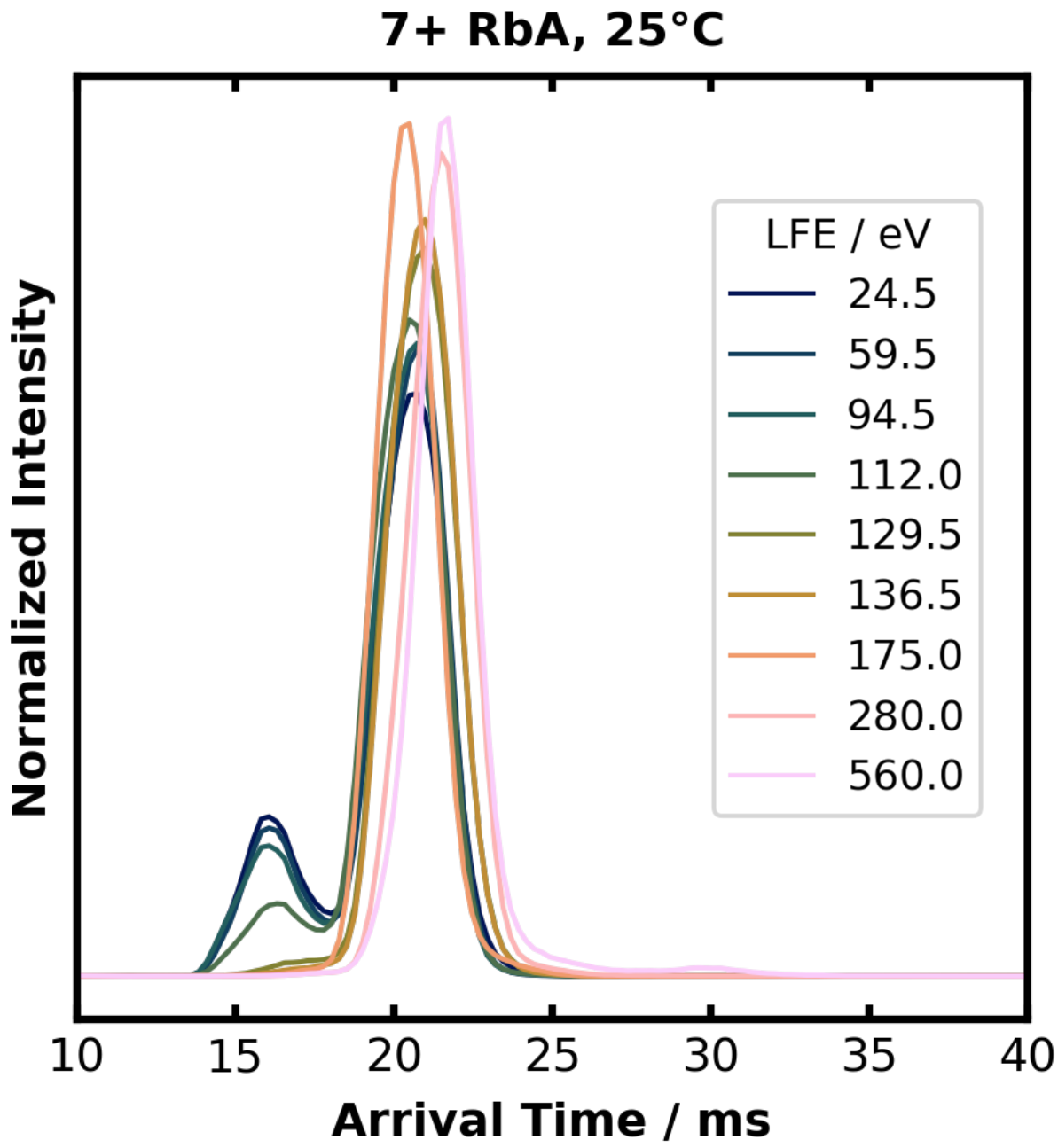


Figure 4.5. CIU type experiment at ambient conditions. Selected laboratory frame energies shown. Note two primary populations, an early arrival time feature and a later arrival time feature that both may contain subpopulations with a slightly earlier or arrival time within the major population grouping.

Using Both ptESI and CIU to Study Protein Unfolding

In Figure 4.8, the 25°C CIU data, panel A, can be seen alongside the novel 90 °C CIU data, panel B. At the higher temperature for lower LFEs, there is a bias towards the early arrival time feature, which is an unexpected result.

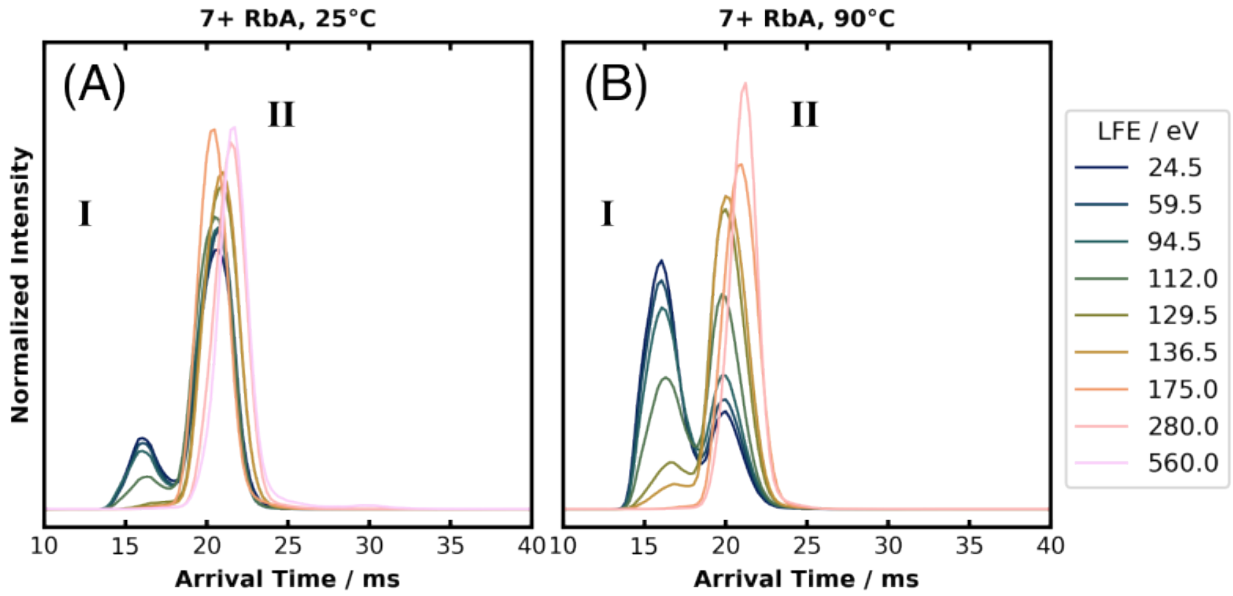


Figure 4.6. Arrival time distributions of RbA 7+ charge state at A) ambient conditions and B) higher temperature conditions at selected laboratory frame energies as indicated in the legend on the right. There are two arrival time features indicated in this graphic as early and late populations. Each arrival time feature may include subpopulations with slightly varied arrival times within the early and late population ranges.

To evaluate this unexpected result, 24.5, 35.0, 112.0, and 560.0 LFE were selected for observation at various solution temperature states, as seen in Figure 4.7A, 4.7B, 4.7C, and 4.7D respectively. As previously noted, the 75 °C and 90 °C data seems to be following a different trend than the other solution temperature states. This trend seems to persist until the highest of LFEs where feature I fully collapses into features II and III. These CIU results point to additional

subpopulations being present within the various arrival time features, especially when both features I and II are populated.

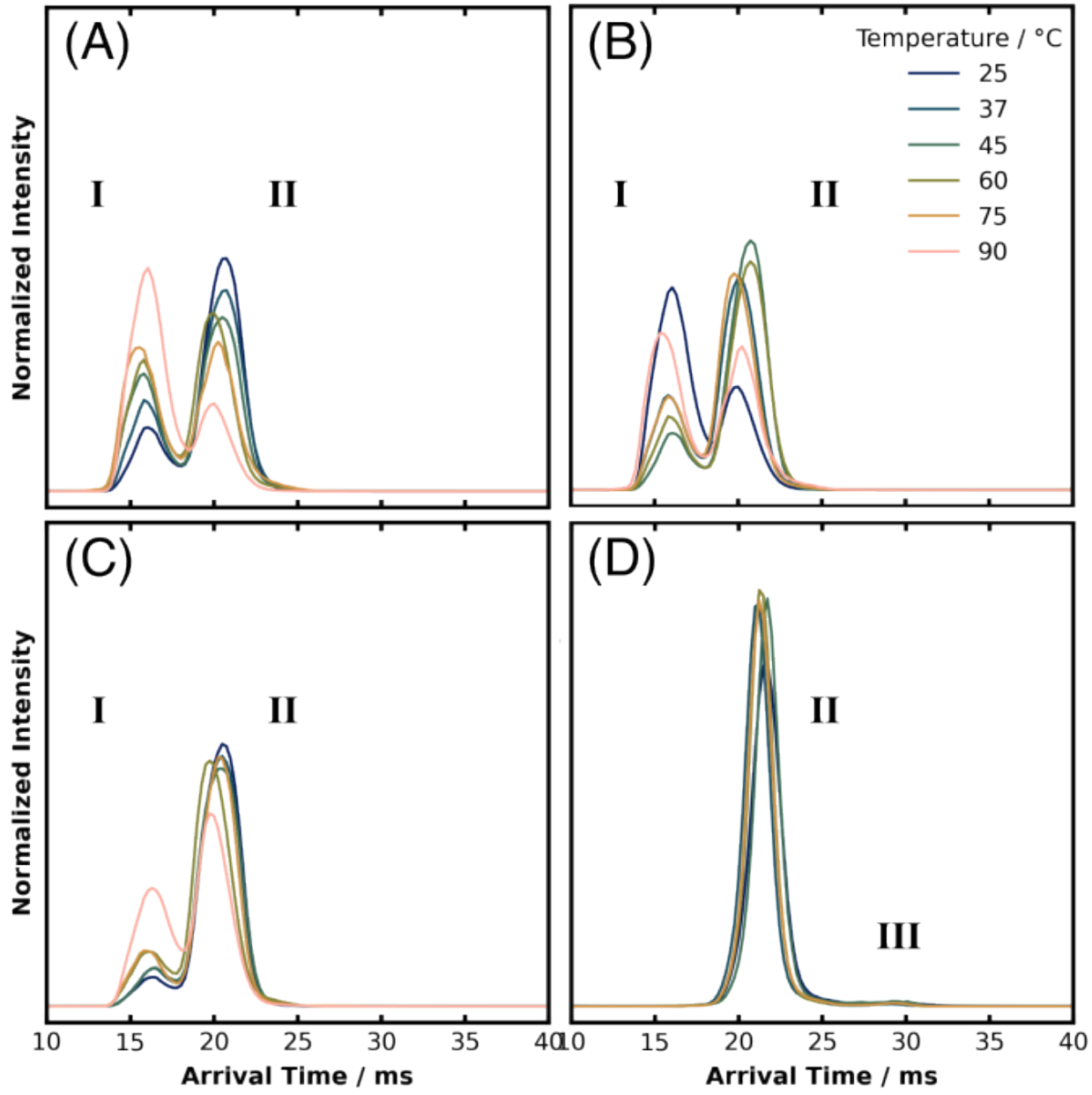


Figure 4.7. Arrival time distributions of RbA 7+ charge state at A) 24.5, B) 35.0, C) 112.0, D) 560.0 LFE (eV) where each colored trace corresponds to the temperatures in the legend in panel B. The features labelled I, II, and III represent early, late, and later arrival time populations that may contain additional subpopulations. Feature I is prior to the inflection point at 17 ms and feature II is after this inflection point but prior to the inflection point at 25 ms and feature III is after the inflection point at 25 ms.

To understand how the subpopulations and populations of the ATDs for the 7+ charge state of RbA were shifting, a relative unfolded population was calculated by leveraging the inflection point between feature I and II, demonstrated by the shading in Figure 4.8. This relative unfolded population was calculated for all temperatures and laboratory frame energies. When this relative unfolded population is plotted against LFEs, a sigmoidal response is seen for 90 °C.

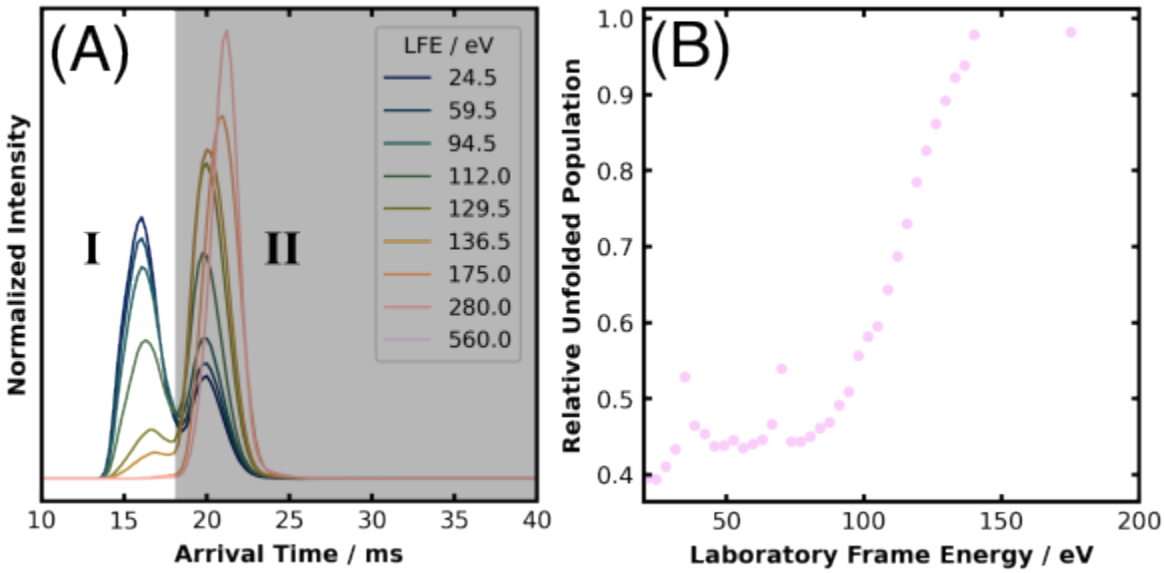


Figure 4.8. A) This panel depicts the arrival time distribution of RbA 7+ charge state at 90 °C at selected activations shown in the colored traces where feature I is an early arrival time feature and feature II is the late arrival time feature. The shaded region, after the inflection point between feature I and II at 17 ms is considered to be unfolded. B) This panel depicts the relative unfolded population compared to the LFE in eV. The relative unfolded population is comprised by the fractional sum of the shaded region in panel A for each LFE.

When comparing this relative unfolded population abundance to LFE for each temperature condition, in general the highest relative unfolded population is populated the lowest temperature condition shown. The relative unfolded population abundance decreases with increasing temperature. This inverse relationship is unexpected.

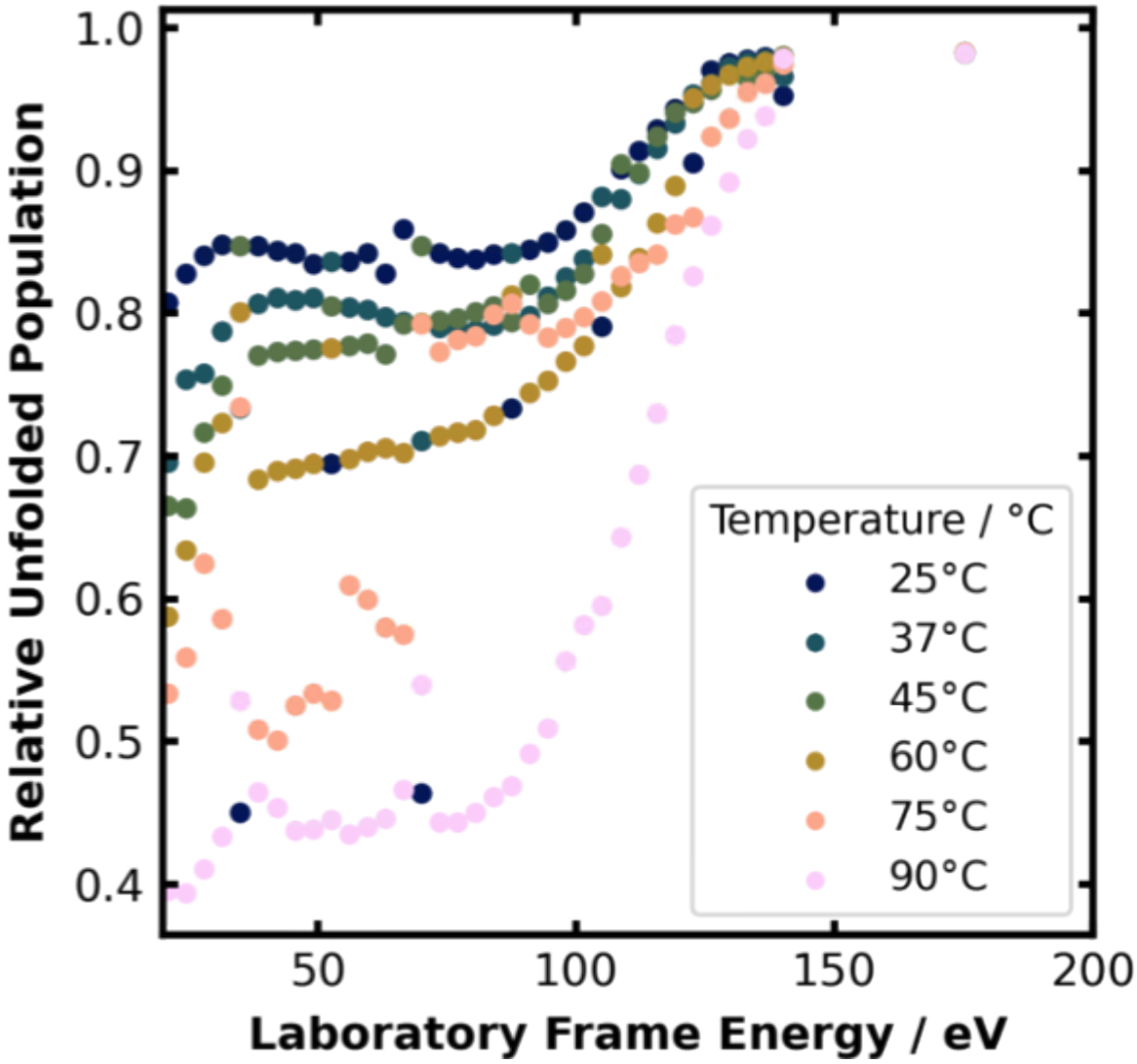


Figure 4.9. For RbA 7+ charge state at various temperatures, the relative unfolded population changes with increasing LFEs.

When evaluating this relative unfolded population for typical ptESI experiments, where only one applied voltage bias in the trap region, only the points in Figure 4.10A are recorded. When utilizing only a CIU workflow, typically at ambient conditions, only the points in Figure 4.10B are recorded. The values of this ptESI-CIU experiment, in Figure 4.10C, showcase that additional energy states may be accessed. The plot in Figure SC2 indicates that there may be a

total of three unfolding events for RbA as indicated by the 90 °C folded and unfolded traces crossing three times.

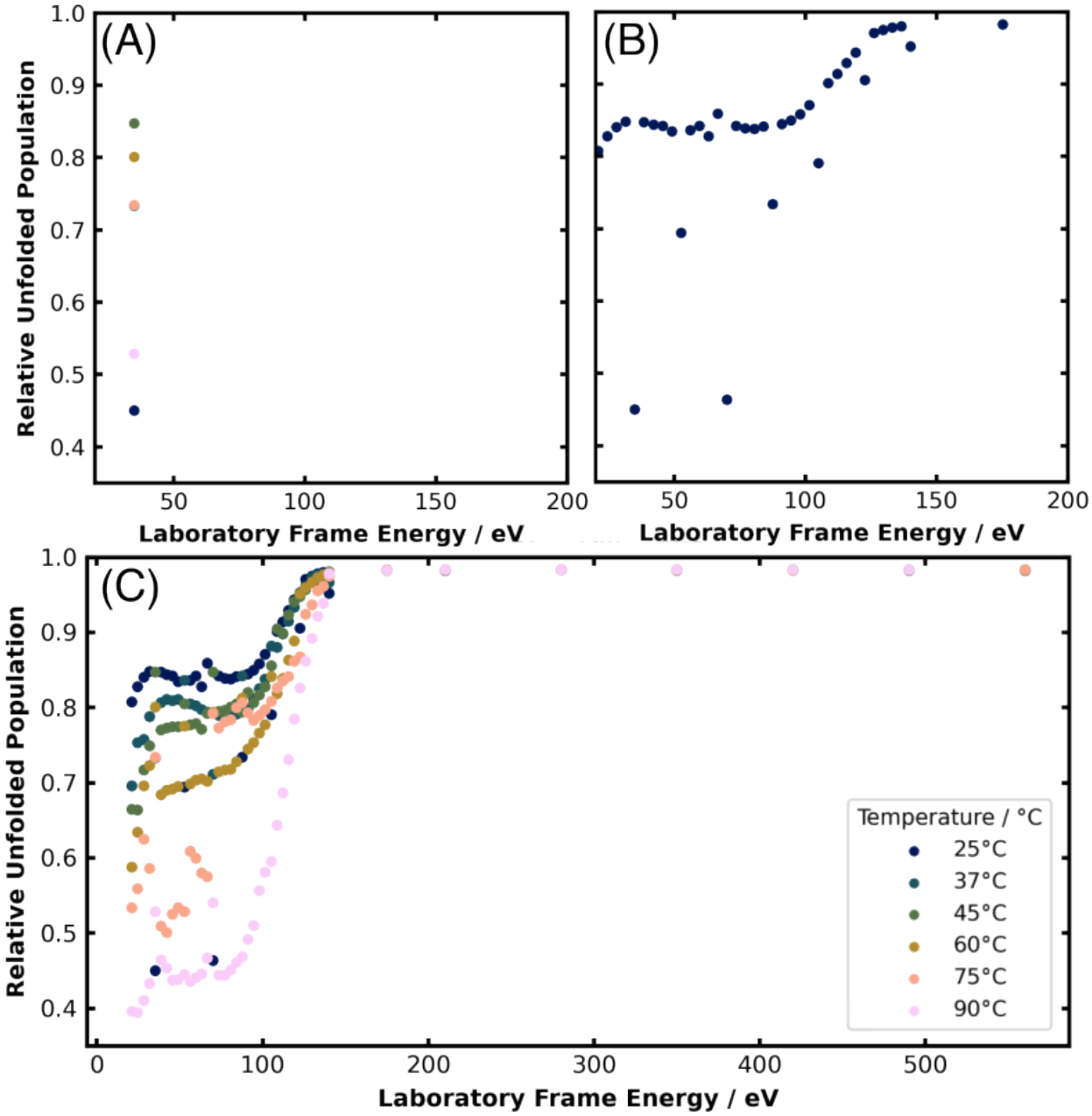


Figure 4.10. A) Unfolded population for ptESI only conditions, B) unfolded population for CIU only conditions, and C) ptESI-CIU for all conditions recorded, this panel expands on Figure 4.9. The temperature legend in this panel applies to all panels.

Conclusions

The additional information provided by ptESI-CIU type experiments can help to uncover additional information about the state of the unfolded population of protein as compared to ptESI type experiment or CIU type experiments alone. This may help to address the variance seen in ptESI type experiments or to probe unfolding not possible with just CIU alone.

Acknowledgements

This work was supported by the National Science Foundation through award 2203513 (Bush) from the Division of Chemistry, with partial co-funding from the Division of Molecular and Cellular Biosciences.

References

(1) Liko, I.; Allison, T. M.; Hopper, J. T.; Robinson, C. V. Mass Spectrometry Guided Structural Biology. *Curr. Opin. Struct. Biol.* **2016**, *40*, 136–144. https://doi.org/10.1016/j.sbi.2016.09.008.

(2) Gabelica, V.; Shvartsburg, A. A.; Afonso, C.; Barran, P.; Benesch, J. L. P.; Bleiholder, C.; Bowers, M. T.; Bilbao, A.; Bush, M. F.; Campbell, J. L.; Campuzano, I. D. G.; Causon, T.; Clowers, B. H.; Creaser, C. S.; De Pauw, E.; Far, J.; Fernandez-Lima, F.; Fjeldsted, J. C.; Giles, K.; Groessl, M.; Hogan, C. J.; Hann, S.; Kim, H. I.; Kurulugama, R. T.; May, J. C.; McLean, J. A.; Pagel, K.; Richardson, K.; Ridgeway, M. E.; Rosu, F.; Sobott, F.; Thalassinos, K.; Valentine, S. J.; Wyttenbach, T. Recommendations for Reporting Ion Mobility Mass Spectrometry Measurements. *Mass Spectrom. Rev.* **2019**, *38* (3), 291–320. https://doi.org/10.1002/mas.21585.

(3) Hyung, S.-J.; Robinson, C. V.; Ruotolo, B. T. Gas-Phase Unfolding and Disassembly Reveals Stability Differences in Ligand-Bound Multiprotein Complexes. *Chem. Biol.* **2009**, *16* (4), 382–390. https://doi.org/10.1016/j.chembiol.2009.02.008.

(4) Dixit, S. M.; Polasky, D. A.; Ruotolo, B. T. Collision Induced Unfolding of Isolated Proteins in the Gas Phase: Past, Present, and Future. *Curr. Opin. Chem. Biol.* **2018**, *42*, 93–100. https://doi.org/10.1016/j.cbpa.2017.11.010.

(5) Hernandez-Alba, O.; Wagner-Rousset, E.; Beck, A.; Cianférani, S. Native Mass Spectrometry, Ion Mobility, and Collision-Induced Unfolding for Conformational Characterization of IgG4 Monoclonal Antibodies. *Anal. Chem.* **2018**, *90* (15), 8865–8872. https://doi.org/10.1021/acs.analchem.8b00912.

(6) Vallejo, D. D.; Kang, J.; Coghlan, J.; Ramírez, C. R.; Polasky, D. A.; Kurulugama, R. T.; Fjeldsted, J. C.; Schwendeman, A. A.; Ruotolo, B. T. Collision-Induced Unfolding Reveals Stability Differences in Infliximab Therapeutics under Native and Heat Stress Conditions. *Anal. Chem.* **2021**, *93* (48), 16166–16174. https://doi.org/10.1021/acs.analchem.1c03946.

(7) Tian, Y.; Lippens, J. L.; Netirojjanakul, C.; Campuzano, I. D. G.; Ruotolo, B. T. Quantitative Collision-induced Unfolding Differentiates Model Antibody–Drug Conjugates. *Protein Sci.* **2019**, *28* (3), 598–608. https://doi.org/10.1002/pro.3560.

(8) Tian, Y.; Han, L.; Buckner, A. C.; Ruotolo, B. T. Collision Induced Unfolding of Intact Antibodies: Rapid Characterization of Disulfide Bonding Patterns, Glycosylation, and Structures. *Anal. Chem.* **2015**, *87* (22), 11509–11515. https://doi.org/10.1021/acs.analchem.5b03291.

(9) Alexander Harrison, J.; Pruška, A.; Oganesyan, I.; Bittner, P.; Zenobi, R. Temperature-Controlled Electrospray Ionization: Recent Progress and Applications. *Chem. – Eur. J.* **2021**, *27* (72), 18015–18028. https://doi.org/10.1002/chem.202102474.

(10) El-Baba, T. J.; Raab, S. A.; Buckley, R. P.; Brown, C. J.; Lutomski, C. A.; Henderson, L. W.; Woodall, D. W.; Shen, J.; Trinidad, J. C.; Niu, H.; Jarrold, M. F.; Russell, D. H.; Laganowsky, A.; Clemmer, D. E. Thermal Analysis of a Mixture of Ribosomal Proteins by vT-ESI-MS: Toward a Parallel Approach for Characterizing the *Stabilitome*. *Anal. Chem.* **2021**, *93* (24), 8484–8492. https://doi.org/10.1021/acs.analchem.1c00772.

(11) Brown, C. J.; Woodall, D. W.; El-Baba, T. J.; Clemmer, D. E. Characterizing Thermal Transitions of IgG with Mass Spectrometry. *J. Am. Soc. Mass Spectrom.* **2019**, *30* (11), 2438–2445. https://doi.org/10.1007/s13361-019-02292-6.

(12) Wang, G.; Abzalimov, R. R.; Kaltashov, I. A. Direct Monitoring of Heat-Stressed Biopolymers with Temperature-Controlled Electrospray Ionization Mass Spectrometry. *Anal. Chem.* **2011**, *83* (8), 2870–2876. https://doi.org/10.1021/ac200441a.

(13) Benesch, J. L. P.; Sobott, F.; Robinson, C. V. Thermal Dissociation of Multimeric Protein Complexes by Using Nanoelectrospray Mass Spectrometry. *Anal. Chem.* **2003**, *75* (10), 2208–2214. https://doi.org/10.1021/ac034132x.

(14) Geels, R. B. J.; Calmat, S.; Heck, A. J. R.; Van Der Vies, S. M.; Heeren, R. M. A. Thermal Activation of the Co-chaperonins GroES and Gp31 Probed by Mass Spectrometry. *Rapid Commun. Mass Spectrom.* **2008**, *22* (22), 3633–3641. https://doi.org/10.1002/rcm.3782.

(15) Cong, X.; Liu, Y.; Liu, W.; Liang, X.; Russell, D. H.; Laganowsky, A. Determining Membrane Protein–Lipid Binding Thermodynamics Using Native Mass Spectrometry. *J. Am. Chem. Soc.* **2016**, *138* (13), 4346–4349. https://doi.org/10.1021/jacs.6b01771.

(16) El-Baba, T. J.; Woodall, D. W.; Raab, S. A.; Fuller, D. R.; Laganowsky, A.; Russell, D. H.; Clemmer, D. E. Melting Proteins: Evidence for Multiple Stable Structures upon Thermal Denaturation of Native Ubiquitin from Ion Mobility Spectrometry-Mass Spectrometry Measurements. *J. Am. Chem. Soc.* **2017**, *139* (18), 6306–6309. https://doi.org/10.1021/jacs.7b02774.

(17) Marchand, A.; Rosu, F.; Zenobi, R.; Gabelica, V. Thermal Denaturation of DNA G-Quadruplexes and Their Complexes with Ligands: Thermodynamic Analysis of the Multiple States Revealed by Mass Spectrometry. *J. Am. Chem. Soc.* **2018**, *140* (39), 12553–12565. https://doi.org/10.1021/jacs.8b07302.

(18) McCabe, J. W.; Shirzadeh, M.; Walker, T. E.; Lin, C.-W.; Jones, B. J.; Wysocki, V. H.; Barondeau, D. P.; Clemmer, D. E.; Laganowsky, A.; Russell, D. H. Variable-Temperature Electrospray Ionization for Temperature-Dependent Folding/Refolding Reactions of Proteins and Ligand Binding. *Anal. Chem.* **2021**, *93* (18), 6924–6931. https://doi.org/10.1021/acs.analchem.1c00870.

(19) Liu, J.; Wang, Y.; Wang, X.; Qin, W.; Li, G. Measuring Protein Unfolding Thermodynamic Stability in One Minute with Digital Temperature Control-Equipped nanoESI-Mass Spectrometry. *Int. J. Mass Spectrom.* **2023**, *494*, 117151. https://doi.org/10.1016/j.ijms.2023.117151.

(20) Constabel, M. A. Programmed Temperature Electrospray Ionization (ptESI) for Thermal Cycling of Proteins. Master of Science, University of Washington, 2024.

(21) Giles, K.; Ujma, J.; Wildgoose, J.; Pringle, S.; Richardson, K.; Langridge, D.; Green, M. A Cyclic Ion Mobility-Mass Spectrometry System. *Anal. Chem.* **2019**, *91* (13), 8564–8573. https://doi.org/10.1021/acs.analchem.9b01838.

(22) Davidson, K. L.; Oberreit, D. R.; Hogan, C. J.; Bush, M. F. Nonspecific Aggregation in Native Electrokinetic Nanoelectrospray Ionization. *Int. J. Mass Spectrom.* **2017**, *420*, 35–42. https://doi.org/10.1016/j.ijms.2016.09.013.

(23) The Inkscape Team. Inkscape. https://inkscape.org/?about-screen=1.

(24) GNU Image Manipulation Program (GIMP). https://www.gimp.org/.

(25) Eller, C. H.; Lomax, J. E.; Raines, R. T. Bovine Brain Ribonuclease Is the Functional Homolog of Human Ribonuclease 1. *J. Biol. Chem.* **2014**, *289* (38), 25996–26006. https://doi.org/10.1074/jbc.M114.566166.

(26) Gozzo, T. A. New Ion Mobility-Mass Spectrometry Tools to Characterize and Differentiate Similar Proteins. PhD, University of Washington, 2023.

(27) Gadzuk-Shea, M. M.; Hubbard, E. E.; Gozzo, T. A.; Bush, M. F. Sample pH Can Drift during Native Mass Spectrometry Experiments: Results from Ratiometric Fluorescence Imaging. *J. Am. Soc. Mass Spectrom.* **2023**, *34* (8), 1675–1684. https://doi.org/10.1021/jasms.3c00147.

(28) Lin, C.-W.; Oney-Hawthorne, S. D.; Kuo, S.-T.; Barondeau, D. P.; Russell, D. H. Mechanistic Insights into IscU Conformation Regulation for Fe–S Cluster Biogenesis Revealed by Variable Temperature Electrospray Ionization Native Ion Mobility Mass Spectrometry. *Biochemistry* **2022**, *61* (23), 2733–2741. https://doi.org/10.1021/acs.biochem.2c00429.

CHAPTER V

Conclusions

To conclude this body of work, I would like to propose a scenario. Let us say that you are interested in onboarding the use of our programmed-temperature electrospray-ionization (ptESI) source to determine which ligand, A or B, improves the stability of the protein therapeutic you've been working on. You've worked with mass spectrometers before and are familiar with their use and understand how to prepare samples for native mass spectrometry (MS). You have some protein of interest sample ready to go and have optimized settings for this protein of interest.

You've just received all the hardware components for the ptESI source and have finished assembling them. You've also received a copy of the acquisition software for the graphical user interface from the Bush Lab. You know that sometimes acquisition software of the mass spectrometer is hosted on a graphical user interface (GUI) that connects on a specific port. This information is available in the url bar at the top of the landing page of the acquisition software of the mass spectrometer, sometimes called a tuning page. You have some experience coding.

Before you start working with the ptESI source, acquire some MS data for your protein of interest with the normal electrospray ionization (ESI) source you normally use in your workflow. If you would like to be prepared, create a sample that has a pH either lower or higher or add formic acid to perform a chemical denaturation of your protein of interest. Run both samples, the native sample and the perturbed sample. Look at both acquisitions and note the mass to charge ratio (m/z) for your protein of interest at native and not native conditions. Identify the charge

states in your protein of interest and their associated *m*/*z* ranges. Record this information. When you get a chance, fill out the protein_charge_state_ranges_template file, that came with the acquisition software in the references folder, with your data. You'll need it later for analysis.

At this point, go ahead and install the acquisition software by running the executable file (.exe) located in the Scripts fold in the venv folder. You'll need to install the DAQ software. Instructions are included in the README.md, which can be read using Notepad. If the software does not launch on its own, then you may need to install the program Anaconda. Once this has been installed, then open an Anaconda Prompt. In this prompt, you'll need to navigate, using the ```cd``` command with the path, to the folder where the software acquisition code is being stored. Once you're in the right folder, install the coding environment by typing ```conda env create -f environment.yml```. Once new outputs stop generating in the Anaconda Prompt, then type ```pip install -r requirements.txt```. After all the new outputs stop generating, then type ```conda activate pt_env```. Congratulations! You can now execute python code. Still in the Anaconda Prompt, type ```flask --app ptESI_control run --host=127.0.0.1 --port=2025``` and then the GUI should load into whatever the last browser you had open or will open in a new window of the default browser.

Once in the GUI, follow the inputs from the top of the page to the bottom of the page. If you're unsure what to use for the parameters, some default settings have been provided, they are greyed out in the input boxes in the parameters section. This first run, upon analysis, will let you know if the solution-phase or gas-phase shifts are adequate for a stability measurement.

If you also received the pt-Analysis code from the Bush Lab, then you may design a python script using the ptESI_data_utils python file in the ptESI_data_utils folder to pick and

choose which classes and functions you can use to perform the analysis necessary for your work alongside the visualizations that showcase your experiment. You can also pick a notebook from the notebooks folder to start development right away with some baseline suggestions. I would suggest specifically making a mass spectra from the total ion chromatogram to compare the ptESI acquisition for the native and non-native sample to see what charge states you are accessing within the ptESI regime. Updating the protein_charge_state_ranges_template file with the accessed charge states would be helpful at this point. I would make a mass spectra from an extracted ion chromatogram for each charge state. You can perform a ptmzlog temperature alignment to get the temperature for each scan in the MS acquisition. You'll need this for the next step.

I would also suggest making an average charge state as it relates to the temperature graph. This graph can compare the stability for ligand A and ligand B. There is a function to perform a sigmoidal fitting to get the midpoint transition for the melted protein, this is the stability measurement for the ptESI system. Once you are happy with your outputs on one ligand, then repeat the same experimental and program conditions for the other ligand, and then you can evaluate the stability both through the charge state as it relates to the temperature visualization and quantifiably by comparing the midpoint transition value for temperature from the sigmoidal fitting.

This scenario describes access to the foundational acquisition and analytical ecosystem for programmed temperature. The new ecosystem allows for flexibility in acquisition by enabling access to temperature program design through simulation alongside running the temperature program. The new ecosystem also enables flexibility in analysis through the design of a fully modular custom Python library. Basic visualizations such as mass spectra from both

total ion chromatograms and extracted ion chromatograms are present alongside more specialized processing that incorporates temperature data alongside *m*/*z* data. There is support for adding access to ion mobility data, multiple mass spectrometry instruments from the same manufacturer, and multiple manufacturers. This pt-Analysis and pt-Control libraries are built to incorporate a variety of needs for multiple laboratories. The design possibilities are endless due to the modular setup of these custom Python libraries. The design possibilities for the pt-Control library are also modular and designed to onboard new hardware seamlessly alongside the existing code base. This means that once the new temperature controller connection is established via additional code that the existing code for the temperature program design and acquisition can still be used. This is equivalent to adding new features instead of having to complete a code refactor to change functionality. The functionality is already established, and custom hardware can be incorporated. In short, this ecosystem was designed to support future work as investigations into protein stability gain complexity over time. The custom Python libraries, pt-Control and pt-Analysis are designed to grow together with the projects, ideas, and hardware developed by researchers.

APPENDIX A

Supplemental Information for Chapter II

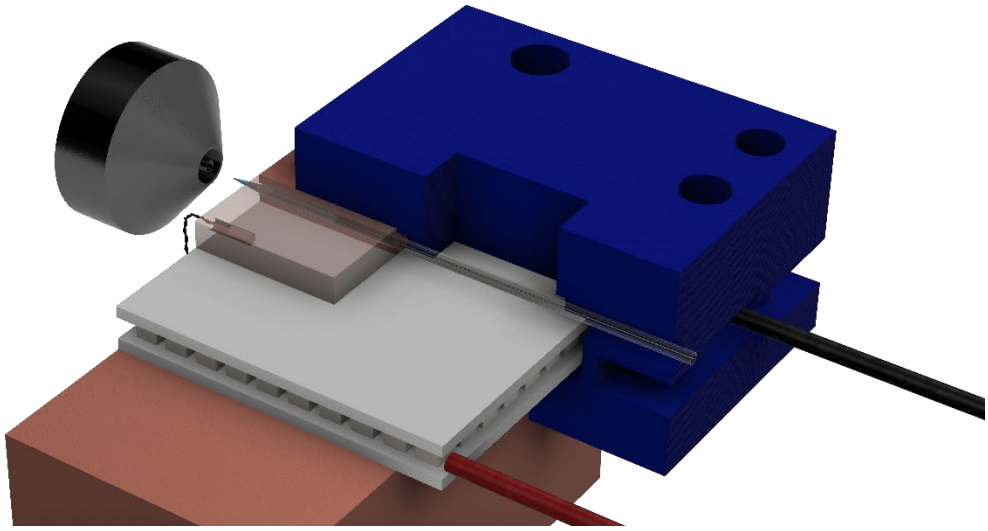


Figure SA1. Render of the ptESI source depicting the solid state heat pump, copper block, embedded thermistor, nESI capillary, copper reservoir, and 3D-printed clamp in front of an instrument inlet.

APPENDIX B

Supplemental Information for Chapter III

Table SB1. Initial mass ranges used for each charge state of RbA

Charge	Low *m/z*	High *m/z*
4+	3419	3425
5+	2735	2740
6+	2279	2284
7+	1954	1958
8+	1710	1713
9+	1520	1523
10+	1368	1371
11+	1244	1247

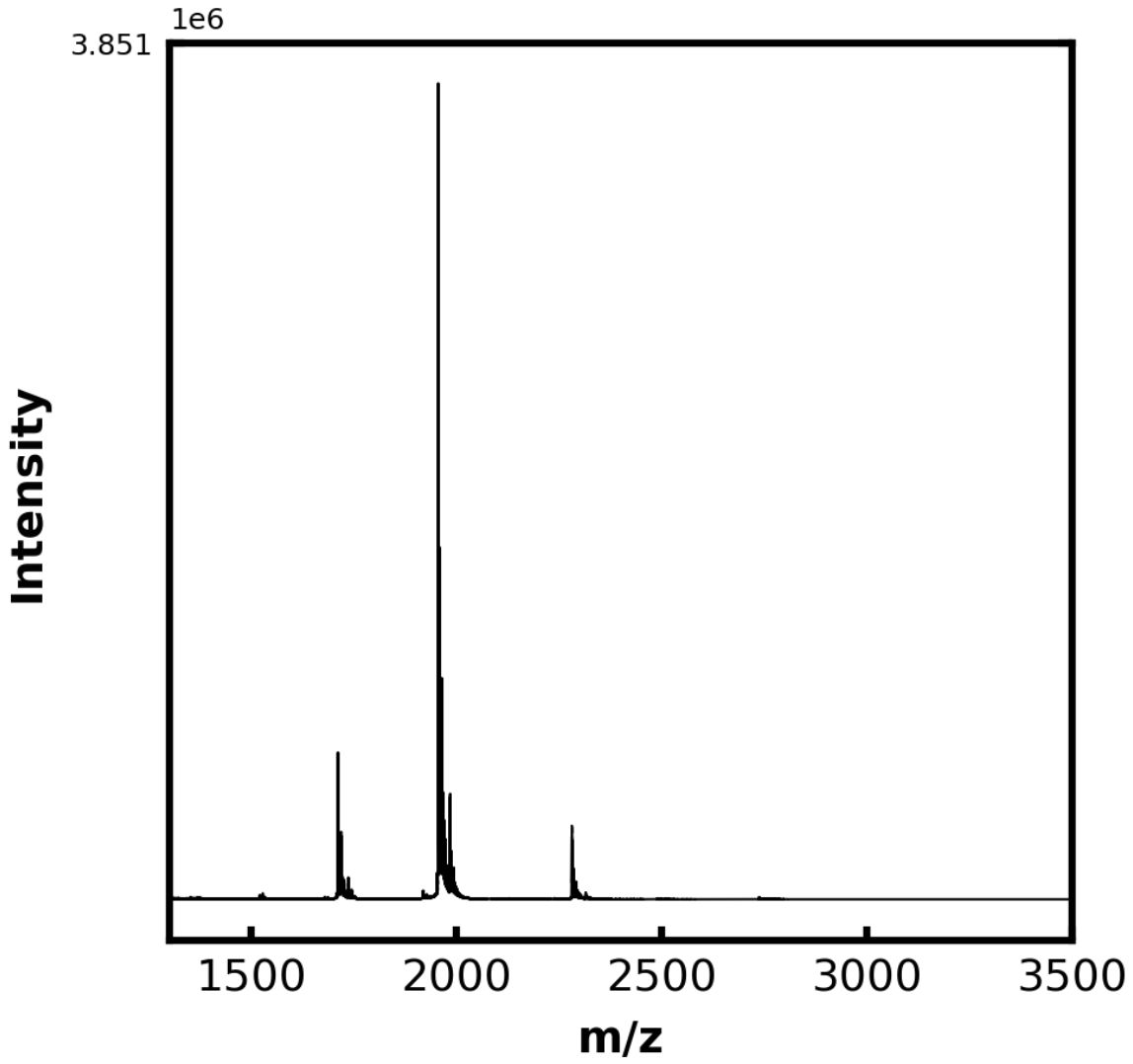


Figure SB1. Mass spectrum for typical ptESI type experiment. All temperatures present, all scans averaged over.

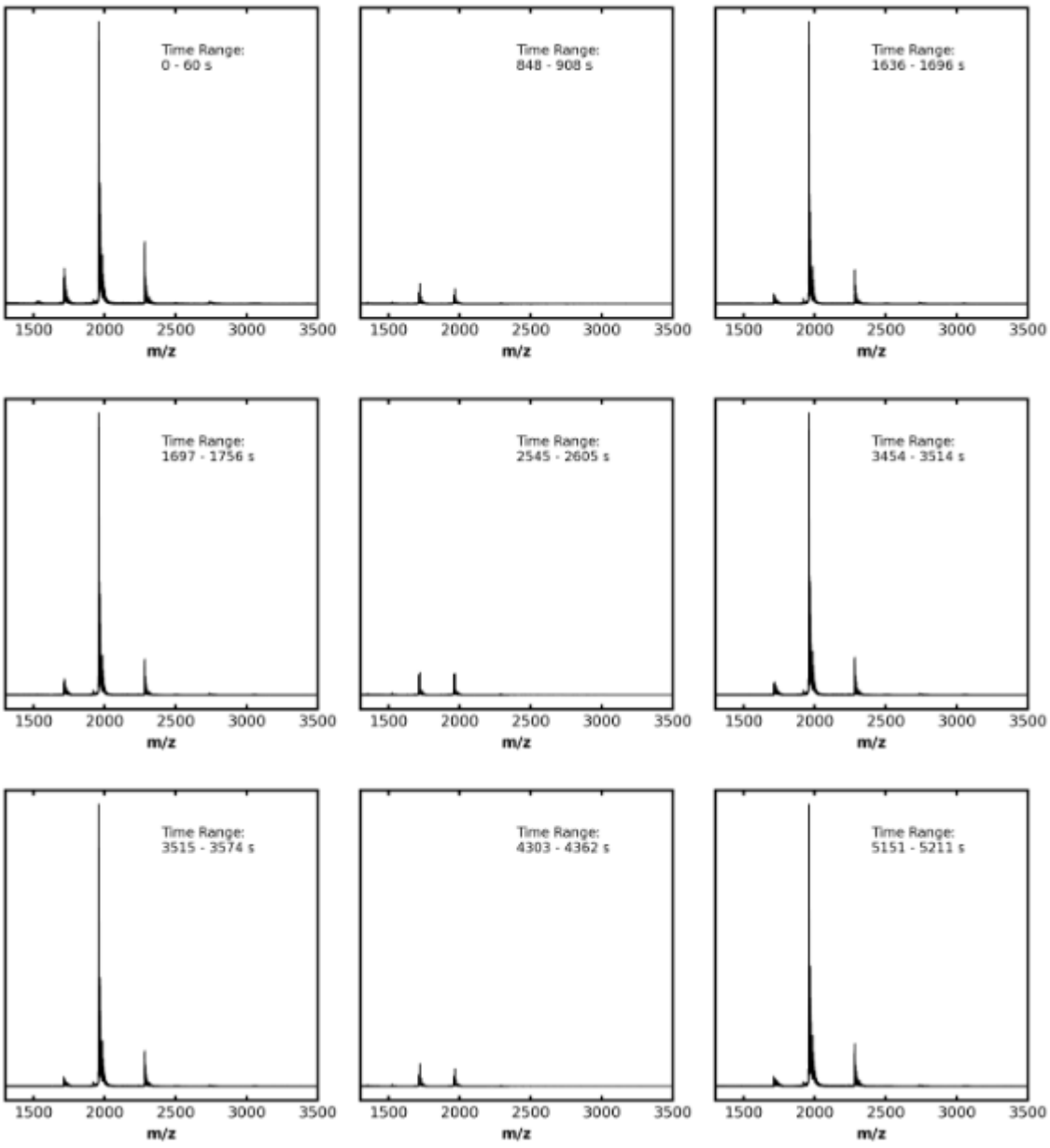


Figure SB2. Mass spectra from the extracted ion chromatograms for 10 °C in first column, 90 °C in second column, 10 °C in right column for cycles 1, 2, and 3 in rows top to bottom respectively for one ptESI experiment with 3 cycles.

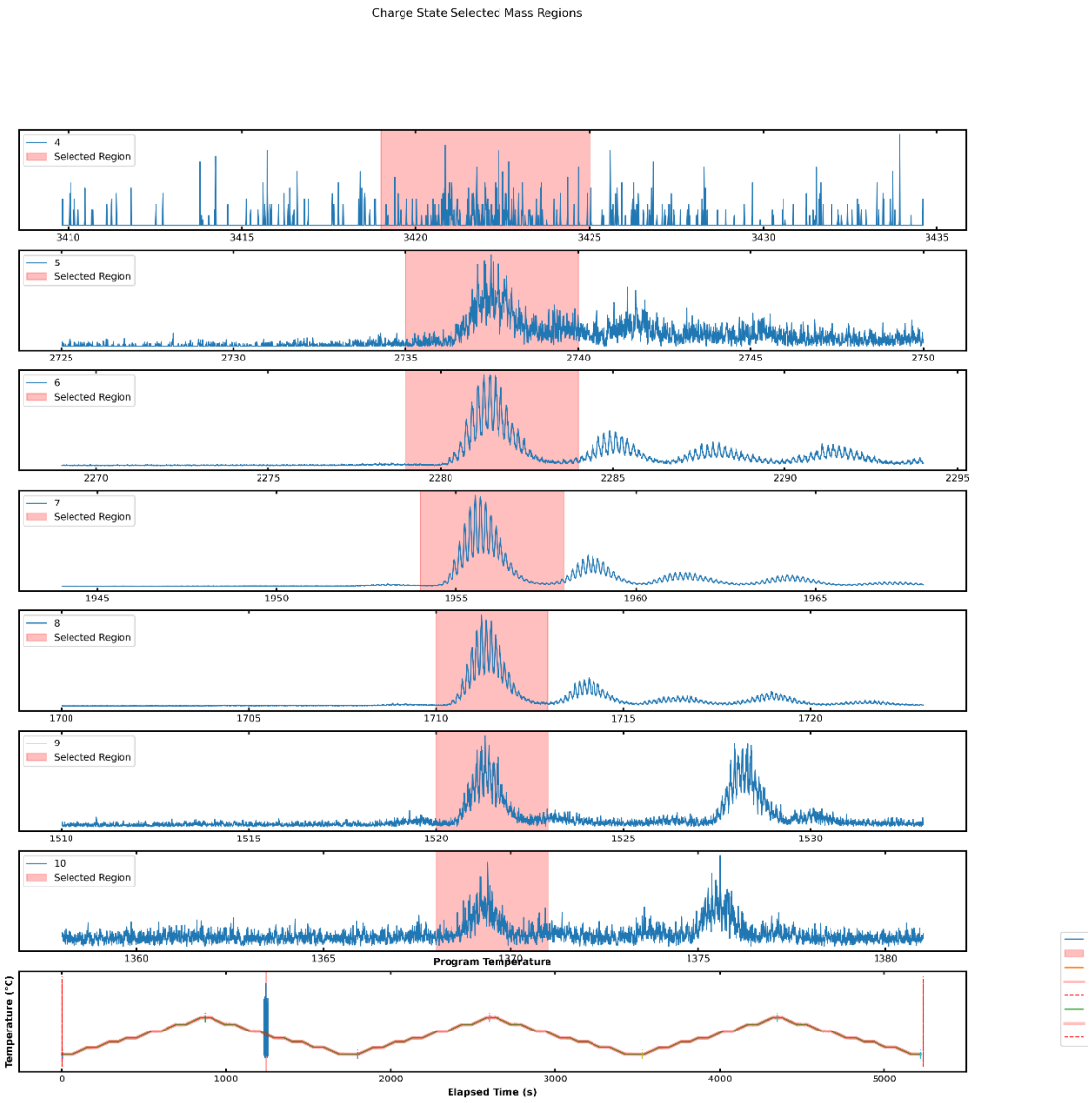


Figure SB3. Example of the mass windows extracted from the total ion chromatogram to make the extracted ion chromatogram for each charge state. This particular acquisition used a temperature cycle of 10 °C to 90°C .

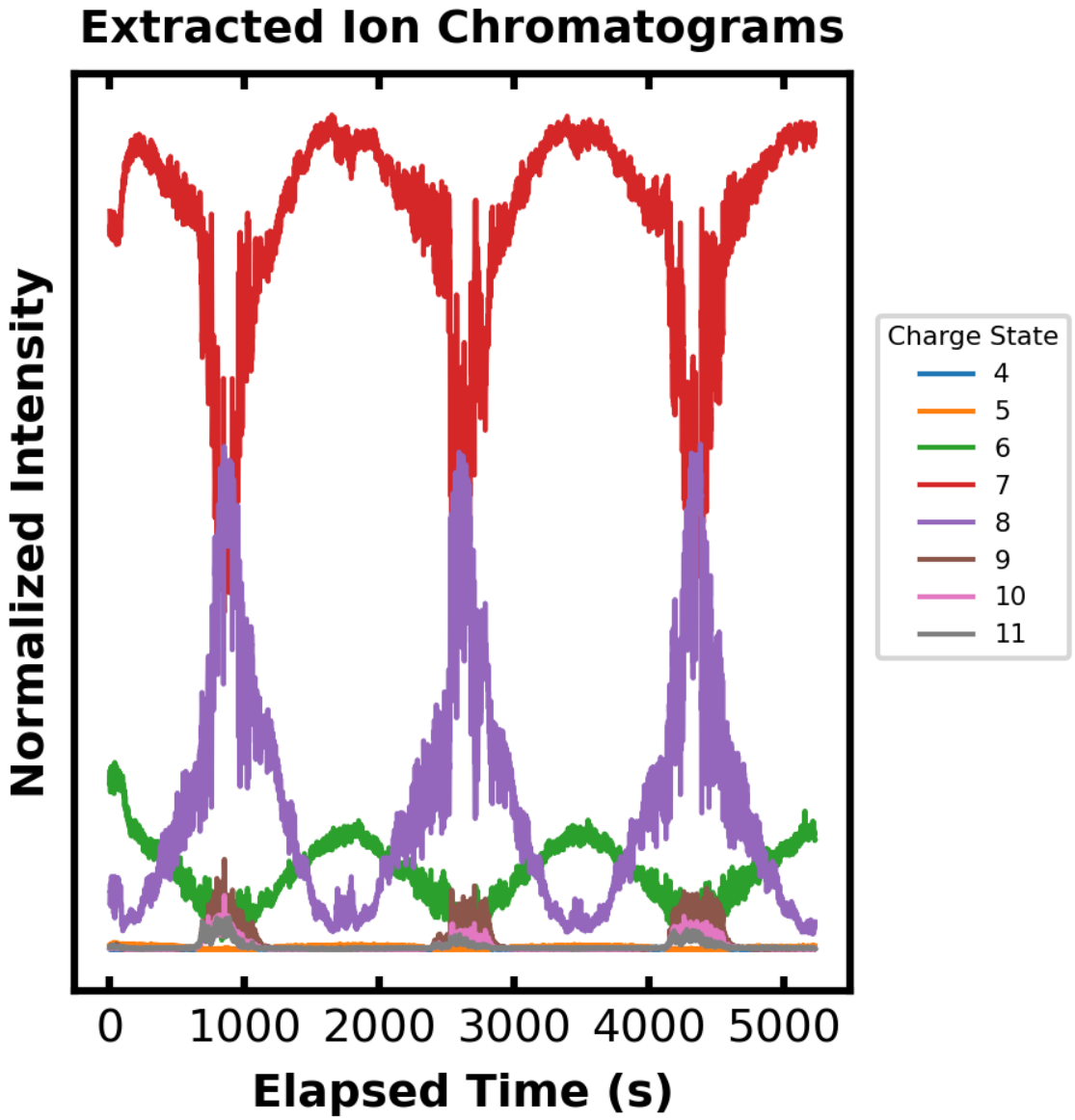


Figure SB4. Extracted ion chromatogram for Ribonuclease A data shown in Figure SB1. It shows the ion chromatogram that has been extracted from the total ion chromatogram for each charge state. The mass ranges for each charge state were set by the analyst, in Table SB1.

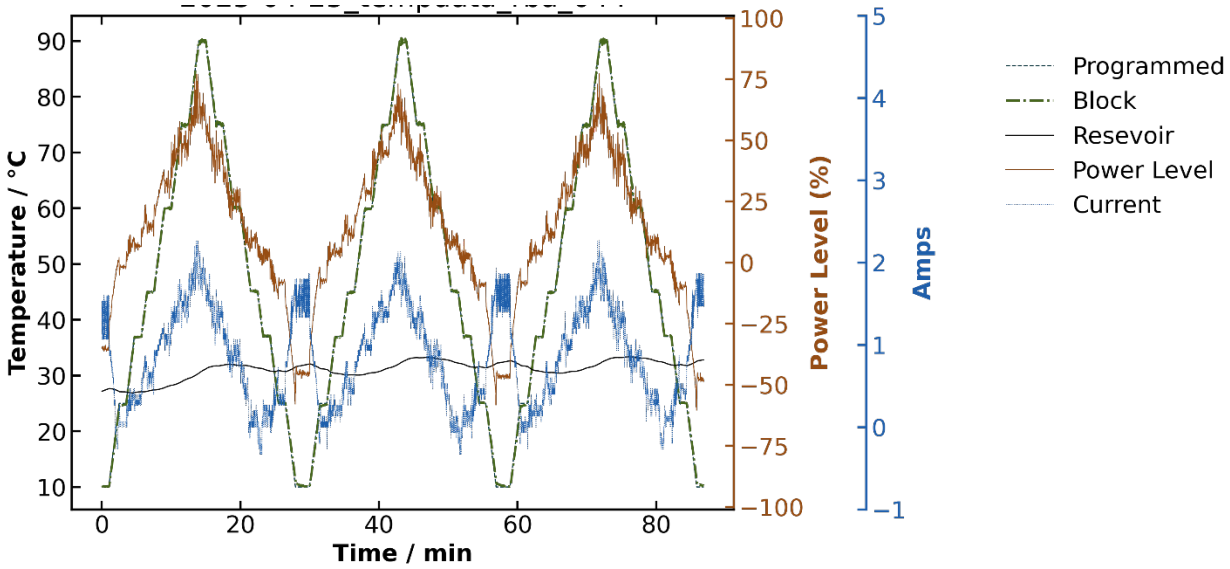


Figure SB5. ptESI experimental readouts showing alignment between programmed temperature and block readings.

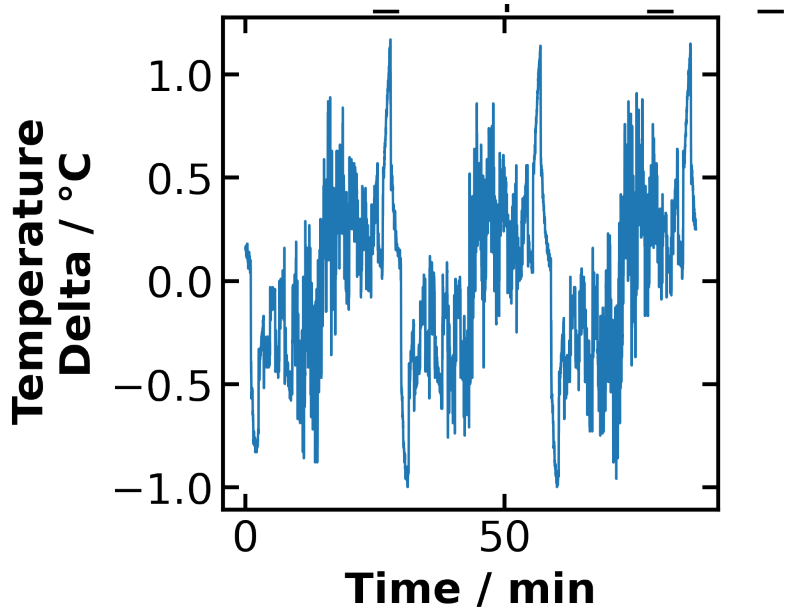


Figure SB6. Temperature differences between the block temperature and the temperature sent to the microcontroller for the duration of the acquisition.

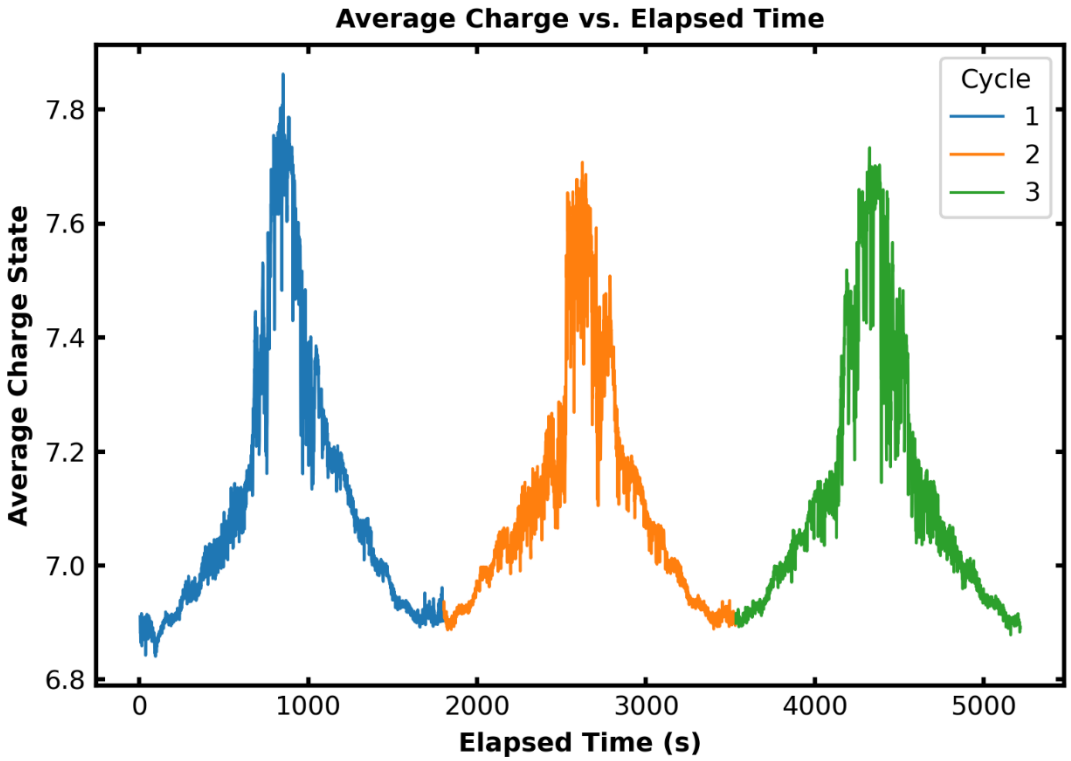
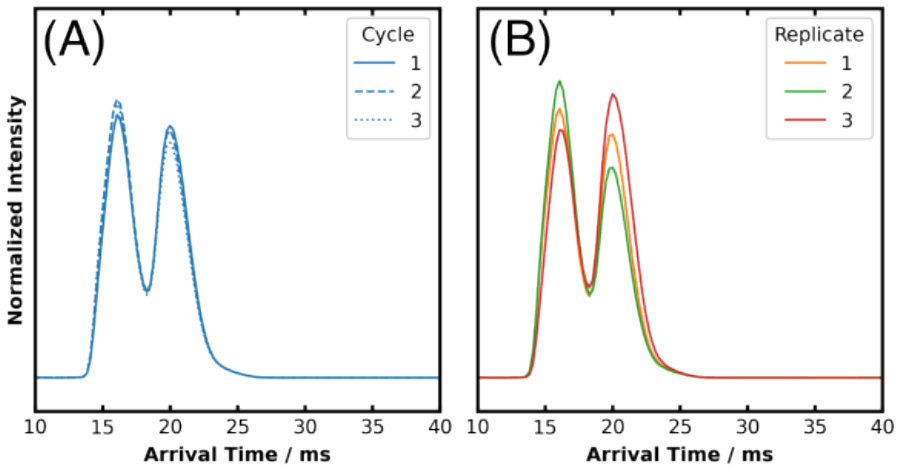
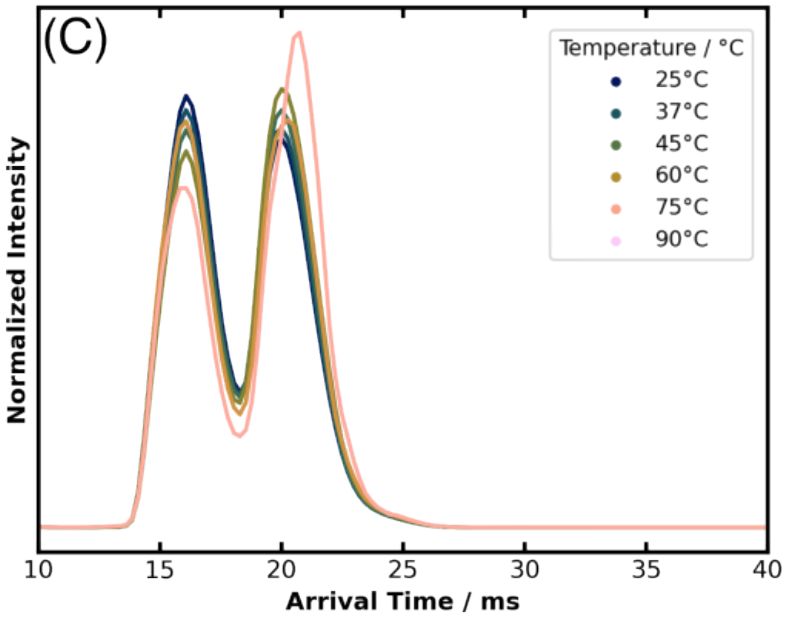


Figure SB7. Average charge state versus elapsed time in seconds, this graphic should be able to uncover what the temperature program was. It is obvious in this graph that there is variation at higher temperatures which correspond ot the elapsed time where temperature is highest.

APPENDIX C

Supplemental Information for Chapter IV


(A)
Cycle
1
2
3
Normalized Intensity
10
15
20
25
30
35
40
Arrival Time / ms
(B)
Replicate
1
2
3
10
15
20
25
30
35
40
Arrival Time / ms


(C)
Temperature / °C
25°C
37°C
45°C
60°C
75°C
90°C
Normalized Intensity
10
15
20
25
30
35
40
Arrival Time / ms

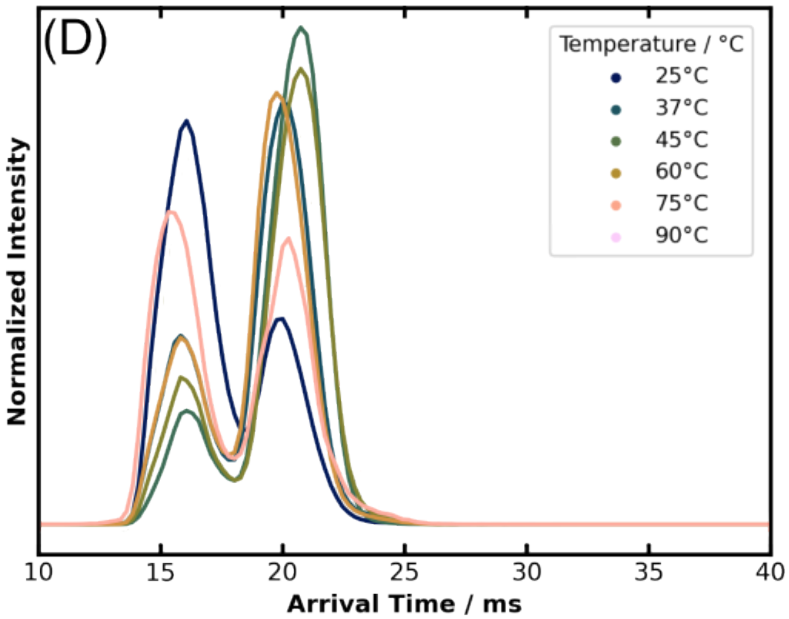

(D)
Temperature / °C
25°C
37°C
45°C
60°C
75°C
90°C
Normalized Intensity
10
15
20
25
30
35
40
Arrival Time / ms

Figure SC1. ptESI and CIU comparison where A) shows the ATDs of different cycles, B) shows the ATDs of different replicates, C) shows the ATDs of extracted temperatures during a ptESI cycling experiment, and D) shows the ATDs of the same temperatures during a CIU experiment at the same activation as the cycling experiment.

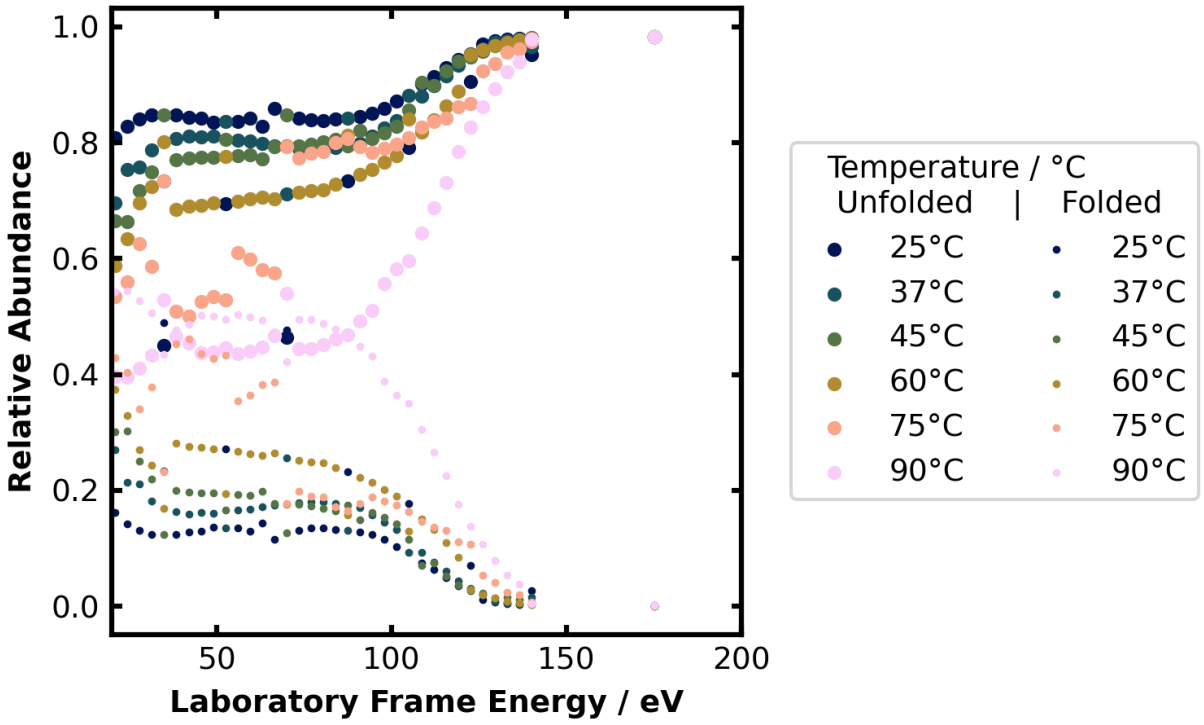


Figure SC2. Relative abundance response to LFEs for various temperature states in the colors indicated in the legend on the right where the folded population is indicated by the smaller points and the unfolded population is indicated by the large points.

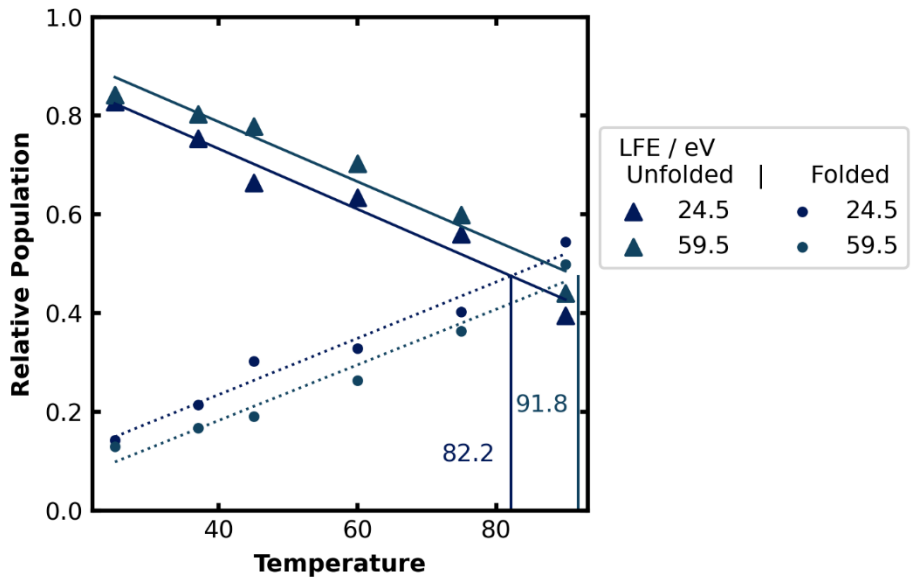


Figure SC3. Relative folded, circle markers, and unfolded, triangle markers, populations for the observed temperatures and the points of intersection between the folded, dashed traces, and unfolded, solid traces for selected laboratory frame energies 24.5. eV in dark blue and 59.5 eV in teal. Only these two lowest selected laboratory frame energies intersected, the temperature at which the folded population began to increase of the unfolded population is printed in text, 82.2 °C and 91.8 °C respectively.