DEVELOPMENT OF A 3D ION TRAP FOR ION/ION REACTIONS AND MASS ANALYSIS INVOLVING HIGH MASS BIOMOLECULAR IONS

by

Kenneth Wayne Lee

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Chemistry

West Lafayette, Indiana

December 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Dr. Scott A. McLuckey, Chair

Department of Chemistry

Dr. Garth J. Simpson

Department of Chemistry

Dr. Paul Wenthold

Department of Chemistry

Dr. Mary J. Wirth

Department of Chemistry

Approved by:

Dr. Christine Hrycyna

Dedicated to my wife Tashina and my parents Wayne and Isabel.

ACKNOWLEDGMENTS

This work was supported by the National Institutes of Health (NIH) under Grant GM R37-45372. Dr. Jeixun Bu and Dr. Eric Dziekonski are acknowledged for initiating our efforts with DIT technology. Dr. James W. Hager of Sciex is acknowledged for helpful discussions and for providing the high voltage conversion dynode. The Purdue Chemistry Department's Jonathan Amy Facility for Chemical Instrumentation staff members – especially Mark Carlsen, Gregory Eakins, and Cathy McIntyre – are acknowledged for their role in developing and building custom electronics for this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

2D	Two-dimensional
3D	Three-dimensional
AC	Alternating current
AcOH	Acetic acid
ASGDI	Atmospheric glow discharge ionization
CID	Collision-induced dissociation
DC	Direct current
DDC	Dipolar direct current
DIO	Digital input/output
ETD	Electron transfer dissociation
ESI	Electrospray ionization
FT	Fourier transform
FT-ICR	Fourier transform ion cyclotron resonance
FWHM	Full width at half maximum
HV	High voltage
IR	Infrared
IRMPD	Infrared multi-photon dissocation
JAFCI	Jonathan Amy Facility for Chemical Instrumentation
KE	Kinetic energy
m/z	Mass-to-charge ratio
MeOH	Methanol
MS	Mass spectrometry
MS^n	Tandem mass spectrometry
nESI	Nano-electrospray ionization
PFO	1H,1H-Perfluoro-1-octanol
QIT	Quadrupole ion trap
RF	Radio frequency
S/N	Signal-to-noise
SID	Surface-induced dissociation
TOF	Time-of-flight
TTL	Transistor-transistor logic
UV	Ultraviolet
UVPD	Ultraviolet Photodissociation

ABSTRACT

Advances in mass spectrometry (MS) instrumentation and techniques have provided approaches for complementing current biochemical research. Native mass spectrometry, which aims to analyze intact biomolecules and biomolecular complexes, has become a powerful tool for identifying and measuring different units of complex structures as well as probing interactions among the different units. Ion traps generally are important in native MS workflows because of their ability to accumulate ions and perform multi-stage analyses including fragmentation, photoreactions, and gas-phase reactions with reagent molecules or ions. Native MS, however, has shortcomings primarily due to the preferred ionization technique, electrospray ionization (ESI). ESI tends to distribute signal from a single analyte among a range of charge states. Additionally, the ions generated from droplets tend to carry adducted molecules and ions proportional to the size of the analyte. For analysis of high mass and heterogeneous biomolecular complexes, these shortcomings lead to wide overlapping charge states for different components that might be difficult to interpret correctly. Charge reduction via gas-phase ion/ion reactions facilitates interpretation of native mass spectra by generating product ions that are well separated in *m/z*. Current sine wave technology limits the upper *m/z* range of ion traps required for stabilizing and measuring high mass ion/ion reaction products. Digital ion trapping (DIT) technology circumvents the voltage limitations of sine wave technology by varying frequency to achieve high *m/z*. The combination of ion/ion reactions and DIT operation facilitates further unique probing reactions such as fragmentation reactions of charge reduced biomolecular complexes via neutral collisions and photoreactions. DIT operation also provides a straightforward approach for isolation of high *m/z* ions using duty cycle modulation to further facilitate analysis of heterogeneous mixtures. This work highlights developments of a home-built 3D ion trap mass spectrometer as a viable native MS platform.

CHAPTER 1. THEORY AND USE OF A 3D QUADRUPOLE ION TRAP MASS SPECTROMETER

1.1 Introduction

The 3D quadrupole ion trap (QIT) is a simple and versatile tool in mass spectrometry (MS). Although mass analysis suffers in terms of accuracy and resolution when compared to high resolution mass analyzers, such as the Fourier transform ion cyclotron resonance (FT-ICR) [1], Orbitrap™ [2], and time-of-flight (TOF) [3] analyzers, QITs readily trap and store ions that can be probed with a variety of techniques prior to mass analysis [4–7]. A unique quality of the 3D QIT vs. the linear QIT is that ions are dynamically trapped and focused in all three spatial dimensions [8,9]. The 3D QIT will trap fewer ions, which decreases signal intensity, but the three-dimensional dynamic trapping facilitates unique experiments such as trapping ions of opposite polarity and laser probing, both which require extra engineering considerations to achieve in a linear QIT [10,11]. Because these experiments can require long-term trapping of a wide range of ion masses and charges, understanding the theory of ion motion in quadrupolar fields highlights limitations and trade-offs that help in experimental design.

1.2 Theory of Ion Motion in a Quadrupolar Field

As with most physics problems, Newton's famous law $F = ma$ gives an appropriate starting point. Because ions have an electrical charge, the force of an electrical field on a charged particle, $F = QE$ will be useful as well. Setting these two equations equal to each other gives:

$$\frac{d^2u}{dt^2} = \frac{ze}{m}\frac{dV}{du} \tag{1.1}$$

where $a = \frac{d^2u}{dt^2}$ is the ion's acceleration in the u dimension, $Q = ze$ is the ion's charge in coulombs, m is the ion's mass, and $E = \frac{dV}{du}$ is the electric field in the u dimension.

The 3D QIT has a ring electrode and two end cap electrodes. Figure 1.1 shows a flattened representation of the rind and end cap electrodes. The trapping field is described with two parameters: radial (r_0), which represents the distance from the trap center towards the ring electrode in two dimensions, and axial (x_0), which represents the distance from the trap center towards either end cap electrode. Standard operation of a QIT puts a time-varying voltage on the

ring electrode while the end caps are held at ground [8]. Because the ions are ideally at the trap's center, there is an effective maximum voltage of $0.5V$ on all electrodes where V is the zero-to-peak amplitude of the time-varying voltage applied to the ring electrode. The voltage at any point in a three-dimensional quadrupolar field is:

$$V(r,x,t) = V(t)\frac{r^2 - 2x^2}{r_0^2 + 2x_0^2} \tag{1.2}$$

where $\frac{r_0^2+2x_0^2}{2}$ is the effective quadrupolar field radius and $V(t)$ is the time-varying voltage with maximum of V. Substitution of Equation (1.1) into (1.2) gives:

$$\frac{d^2r}{dt^2} = \frac{2ze}{m(r_0^2 + 2x_0^2)}V(t)r \qquad \frac{d^2x}{dt^2} = -\frac{4ze}{m(r_0^2 + 2x_0^2)}V(t)x \tag{1.3}$$

Depending on the nature of $V(t)$, Equation (1.3) has an analytical solution that describes the motion of the trapped ion in both the radial and axial dimensions. Stable trapping of ions is accomplished by using periodic waveforms; therefore, analysis of Equation (1.3) is possible by solving the differential equation over one period of $V(t)$, rather than all time. A simple change of variables using $\xi = \frac{\Omega t}{2}$ generalizes Equation (1.3) to unitless time giving:

$$\frac{d^2r}{d\xi^2} = \frac{8ze}{m\Omega^2(r_0^2 + 2x_0^2)}V(\xi)r \qquad \frac{d^2x}{d\xi^2} = -\frac{16ze}{m\Omega^2(r_0^2 + 2x_0^2)}V(\xi)x \tag{1.4}$$

where $\boldsymbol{\Omega}$ is the frequency of the periodic waveform in radians per second. One period of the voltage is now $\xi = [0, \pi]$, which is independent of frequency, rather than $t = \left[0, \frac{2\pi}{\Omega}\right]$.

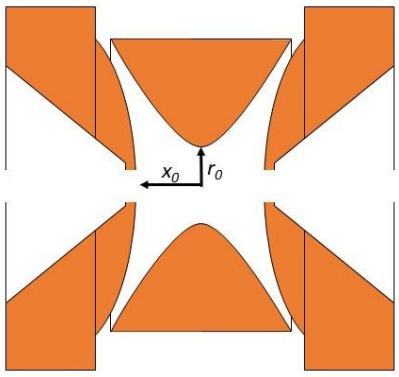

Figure 1.1. Flattened representation of a 3D quadrupole ion trap with a central ring electrode and two end cap electrodes. The trap field dimensions are the center-to-ring electrode distance (r_0) and the center-to-end cap distance (x_0).

1.2.1 Stable Ion Motion in a Sine Wave Quadrupolar Field

A generic sine wave potential is:

$$V(t) = U + V\cos(\Omega t) \tag{1.5}$$

where U is the DC offset of the sine wave, V is the zero-to-peak amplitude, and Ω is the frequency in radians per second. Using unitless time, this becomes:

$$V(\xi) = U + V\cos(2\xi) \tag{1.6}$$

Substitution of Equation (1.6) into the axial part of Equation (1.4) gives:

$$\frac{d^2x}{d\xi^2} + \frac{16ze}{m\Omega^2(r_0^2 + 2x_0^2)}[U + V\cos(2\xi)]x = 0 \tag{1.7}$$

When compared to the canonical Mathieu equation given by:

$$\frac{d^2u}{d\xi^2} + [a_u - 2q_u\cos(2\xi)]u = 0 \tag{1.8}$$

the two parameters a_x and q_x are proportional to the DC offset, U, and the amplitude, V, illustrated by the following:

$$a_x = \frac{16zeU}{m\Omega^2(r_0^2 + 2x_0^2)} \qquad q_x = -\frac{8zeV}{m\Omega^2(r_0^2 + 2x_0^2)} \tag{1.9}$$

The same procedure can relate the radial part of Equation (1.4) to parameters a_r and q_r given by:

$$a_r = -\frac{8zeU}{m\Omega^2(r_0^2 + 2x_0^2)} \qquad q_r = \frac{4zeV}{m\Omega^2(r_0^2 + 2x_0^2)} \tag{1.10}$$

The solutions of the Mathieu equation are either stable or unstable. The combinations of a_x and q_x that give stable solutions inform the user which combinations of U, V, and Ω will stably trap an ion of mass m and unit charge z in the axial dimension. The same is true for combinations of a_r and q_r in the radial dimension. Figure 1.2 is a plot of a_x vs. q_x with the radial stability region colored in red and the axial stability region colored in blue. Both dimensions can be plotted together by noting the relationships between the Mathieu parameters for the two dimensions, namely that $a_r = -2a_x$ and $q_r = -2q_x$.

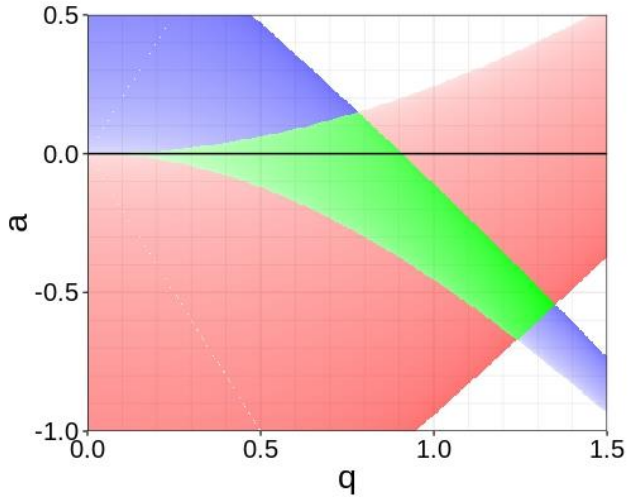

Figure 1.2. Mathieu stability diagram for a 3D ion trap operated with a sine wave. Radial stability is in red and axial stability is in blue.

1.2.2 Stable Ion Motion in a Square Wave Quadrupolar Field

A non-sinusoidal trapping potential cannot be addressed using the solutions to the Mathieu equation; however, an exact solution to the differential equation (1.4) using a square wave potential can be found by solving the equation in a piecewise approach [12]. A generic square wave potential can be given by:

$$V(t) = \begin{cases} V_1, & 0 < t < Td \\ V_2, & Td < t < T \end{cases} \tag{1.11}$$

where V_1 and V_2 are the two rail voltages of the square wave, d is the duty cycle of the square wave (i.e., the percentage of the period spent at V_1 so that Td is the time in seconds spent at V_1), and T is the square wave period is seconds. Changing the equation to unitless time gives:

$$V(\xi) = \begin{cases} V_1, & 0 < \xi < \pi d \\ V_2, & \pi d < \xi < \pi \end{cases} \tag{1.12}$$

Because $V(\xi)$ is constant during each portion of the square wave, Equation (1.4) has an analytical piecewise solution. It is useful to introduce a new variable:

$$f(\xi) = \begin{cases} f_1 = -\dfrac{16Q}{m\Omega^2(r_0^2 + 2z_0^2)}V_1, & 0 < \xi < \pi d \\ f_2 = -\dfrac{16Q}{m\Omega^2(r_0^2 + 2z_0^2)}V_2, & \pi d < \xi < \pi \end{cases} \tag{1.13}$$

so that substitution of 1.12 into the axial portion of 1.4 gives:

$$\begin{cases} \dfrac{d^2x}{d\xi^2} + f_1 x = 0, \;\; 0 < \xi < \pi d \\ \dfrac{d^2x}{d\xi^2} + f_2 x = 0, \;\; \pi d < \xi < \pi \end{cases} \tag{1.14}$$

Each part of the piecewise differential equation has a solution:

$$x(\xi) = \begin{cases} x_0 \cos\left(\sqrt{f_k}\xi\right) + \dfrac{x_0'}{\sqrt{f_k}} \sin\left(\sqrt{f_k}\xi\right), & f_k > 0 \\ x_0 \cosh\left(\sqrt{-f_k}\xi\right) + \dfrac{x_0'}{\sqrt{-f_k}} \sinh\left(\sqrt{-f_k}\xi\right), & f_k < 0 \end{cases}, \;\; k = 1,2 \tag{1.15}$$

where x_0 and x_0' are the initial position and velocity of the ion, respectively. Equation (1.15) can be rewritten as a matrix using an expression for the ion velocity:

$$x'(\xi) = \begin{cases} -x_0\sqrt{f_k} \sin\left(\sqrt{f_k}\xi\right) + x_0' \cos\left(\sqrt{f_k}\xi\right), & f_k > 0 \\ x_0\sqrt{-f_k} \sinh\left(\sqrt{-f_k}\xi\right) + x_0' \cosh\left(\sqrt{-f_k}\xi\right), & f_k < 0 \end{cases}, \;\; k = 1,2 \tag{1.16}$$

The matrix expression is then:

$$\begin{bmatrix} x(\xi) \\ x'(\xi) \end{bmatrix} = \begin{bmatrix} \cos\left(\sqrt{f_k}\xi\right) & \dfrac{1}{\sqrt{f_k}} \sin\left(\sqrt{f_k}\xi\right) \\ -\sqrt{f_k} \sin\left(\sqrt{f_k}\xi\right) & \cos\left(\sqrt{f_k}\xi\right) \end{bmatrix} \begin{bmatrix} x_0 \\ x_0' \end{bmatrix}, \;\; f_k > 0 \tag{1.17}$$

$$\begin{bmatrix} x(\xi) \\ x'(\xi) \end{bmatrix} = \begin{bmatrix} \cosh\left(\sqrt{-f_k}\xi\right) & \dfrac{1}{\sqrt{-f_k}} \sinh\left(\sqrt{-f_k}\xi\right) \\ \sqrt{-f_k} \sinh\left(\sqrt{-f_k}\xi\right) & \cosh\left(\sqrt{-f_k}\xi\right) \end{bmatrix} \begin{bmatrix} x_0 \\ x_0' \end{bmatrix}, \;\; f_k < 0 \tag{1.18}$$

Typical operation of an ion trap will use $V_1 > 0$ and $V_2 < 0$, such that Equation (1.18) will apply to the first part of the square wave period and Equation (1.17) will apply to the second part. Because the final position and velocity after the first part of the square wave correspond to the initial position and velocity of the second part of the square wave, the final position and velocity after one full period of the square wave can be determined by a product of matrices:

$$\begin{bmatrix} x(\pi) \\ x'(\pi) \end{bmatrix} = M \begin{bmatrix} x_0 \\ x_0' \end{bmatrix} \tag{1.19}$$

$$M = \begin{bmatrix} \cos\left(\sqrt{f_2}\pi(1-d)\right) & \dfrac{1}{\sqrt{f_2}} \sin\left(\sqrt{f_2}\pi(1-d)\right) \\ -\sqrt{f_2} \sin\left(\sqrt{f_2}\pi(1-d)\right) & \cos\left(\sqrt{f_2}\pi(1-d)\right) \end{bmatrix} \begin{bmatrix} \cosh\left(\sqrt{-f_1}\pi d\right) & \dfrac{1}{\sqrt{-f_1}} \sinh\left(\sqrt{-f_1}\pi d\right) \\ \sqrt{-f_1} \sinh\left(\sqrt{-f_1}\pi d\right) & \cosh\left(\sqrt{-f_1}\pi d\right) \end{bmatrix} \tag{1.20}$$

Equation (1.19) will indicate ion stability over one period, but to determine stability over *n* periods, the *n*th power of the matrix *M* must be determined. Careful algebra will show that $\det M = 1$, which implies that *M* has exactly two eigenvalues and eigenvectors, such that:

$$Mm = \lambda m \tag{1.21}$$

where m represents an eigenvector and λ represents the corresponding eigenvalue. From the properties of eigenvectors, the initial ion vector (i.e., the initial position and velocity) is a linear combination of the two eigenvectors, namely:

$$\begin{bmatrix} x_0 \\ x_0' \end{bmatrix} = C_1 m_1 + C_2 m_2 \tag{1.22}$$

Combining Equations (1.19) and (1.22) gives:

$$\begin{bmatrix} x(\pi) \\ x'(\pi) \end{bmatrix} = M(C_1 m_1 + C_2 m_2) = C_1 \lambda_1 m_1 + C_2 \lambda_2 m_2 \tag{1.23}$$

The position and velocity can now be easily expressed in terms of the eigenvalues and eigenvectors of M at the end of n periods as:

$$\begin{bmatrix} x(n\pi) \\ x'(n\pi) \end{bmatrix} = M^n(C_1 m_1 + C_2 m_2) = C_1 \lambda_1^n m_1 + C_2 \lambda_2^n m_2 \tag{1.24}$$

Inspection of Equation (1.24) shows that the condition for ion stability is:

$$|\lambda_{1,2}| \leq 1 \tag{1.25}$$

The eigenvalue problem of Equation (1.21) can be solved to express the condition for ion stability in terms of the matrix M. Rewriting Equation (1.21) with matrix and vector elements gives:

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \lambda \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} \tag{1.26}$$

$$\begin{cases} (M_{11} - \lambda)m_1 + M_{12}m_2 = 0 \\ M_{21}m_1 + (M_{22} - \lambda)m_2 = 0 \end{cases} \tag{1.27}$$

The above system of equations can be written as a matrix whose determinant must be zero for the system of equations to have nonzero solutions:

$$\det \begin{bmatrix} M_{11} - \lambda & M_{12} \\ M_{21} & M_{22} - \lambda \end{bmatrix} = \lambda^2 - (M_{11} + M_{22})\lambda + M_{11}M_{22} - M_{12}M_{21} = 0 \tag{1.28}$$

Because the determinant of M (i.e., $M_{11}M_{22} - M_{12}M_{21}$) is one, the two possible eigenvalues are functions of the trace of M, namely:

$$\lambda = \frac{M_{11} + M_{22}}{2} \pm i\sqrt{1 - \left(\frac{M_{11} + M_{22}}{2}\right)^2} \tag{1.29}$$

To hold the condition established in Equation (1.25), the following must be true:

$$\left|\frac{M_{11} + M_{22}}{2}\right| \leq 1 \tag{1.30}$$

Thus, to determine ion stability in a square wave potential, the matrix M is calculated using Equation (1.20) for both the radial and axial dimensions. If the absolute values of both traces are less than or equal to two, the ion will be stable over n periods of the square wave.

While this matrix approach gives exact solutions for waveforms that have periods of constant voltage (e.g., square waves), it is also useful for approximating solutions for waveforms that have no periods of constant voltage (e.g., sine waves). The waveform $V(\xi)$ is simply expressed as a piecewise function of k steps of constant voltage. As k increases, the accuracy of the matrix solutions will also increase. With a large enough k, this approach gives identical results to the standard method of solving ion stability in sine wave potentials [12].

1.2.3 Comparison of Ion Stability in Sine Wave and Square Wave Fields

A direct comparison of sine wave and square wave trapping is not straightforward because of the unique parameterization of each. Whereas a sine wave is easily reduced to a DC offset, amplitude, and frequency, a square wave has the additional parameter of duty cycle. From the definition of the parameter f in Equation (1.13), it is natural to see a connection to the standard Mathieu parameters a and q. Two common definitions of a and q in terms of f are in use [13]. The first attempts to define a as a weighted DC offset and q as a weighted amplitude:

$$a = f_1 d + f_2(1-d) \qquad q = (f_1 - f_2)d(1-d) \tag{1.31}$$

The second more simply defines a and q as an unweighted average potential and amplitude, respectively:

$$a = \frac{f_1 + f_2}{2} \qquad q = \frac{f_1 - f_2}{4} \tag{1.32}$$

The division by four in the definition for q comes from the fact that the peak-to-peak amplitude of a sine wave is always equal to $4q$ using the standard Mathieu parameters. Figure 1.3 compares the stability regions for a (a) 40%, (b) 50%, and (c) 60% duty cycle square wave using Equation (1.31) vs. a (d) 40%, (e) 50%, and (f) 60% duty cycle square wave using Equation (1.32). The first definition is more useful when comparison sine and square wave stability diagrams, because a and q for a square wave will more closely correspond to the standard Mathieu parameters for a sine wave. The second definition is more practical, though, because a and q are easily calculated. However, both definitions have two weaknesses: the stability diagram changes with duty cycle, and both parameters are dependent on the same physical characteristics of the square wave.

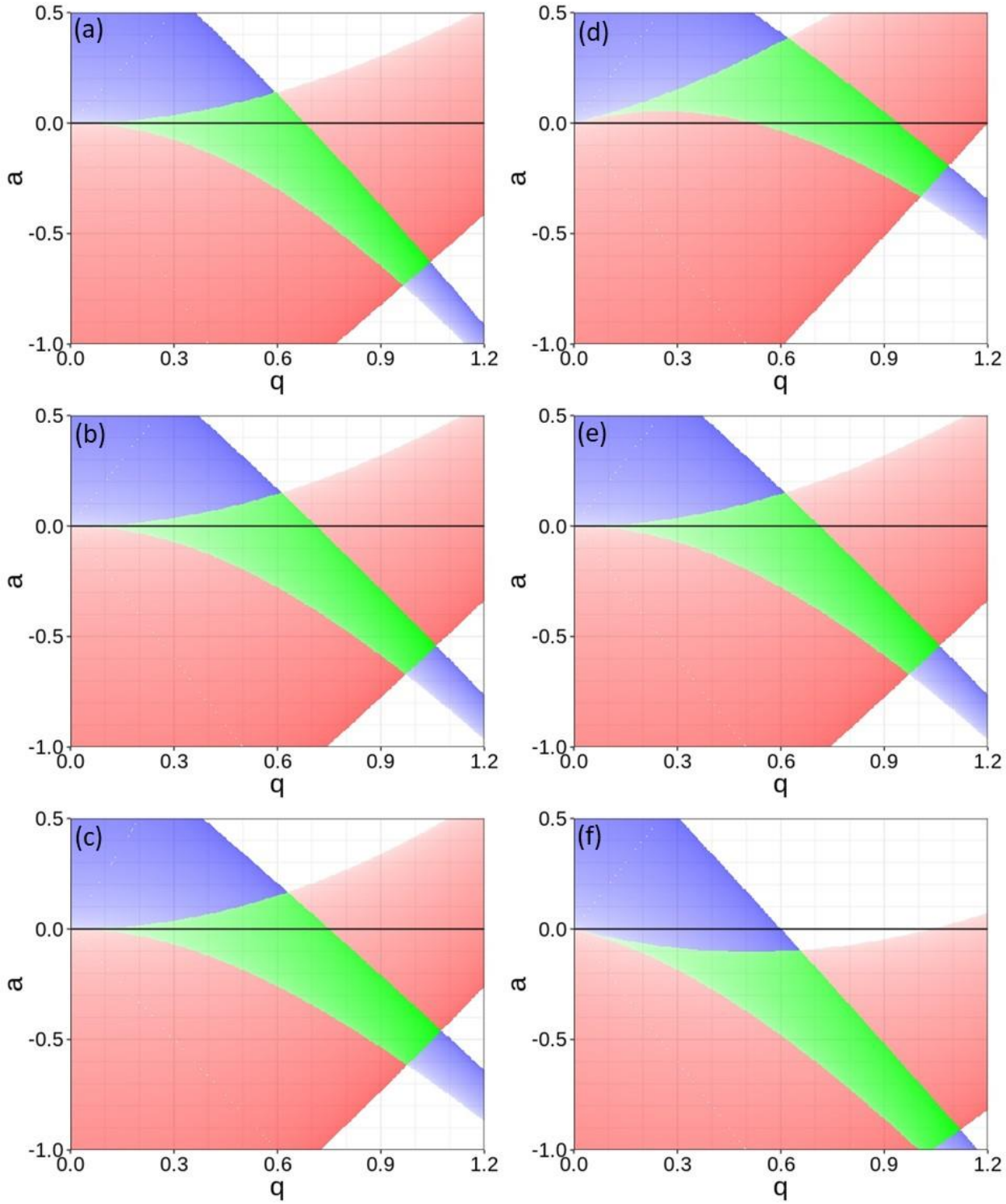

Figure 1.3. Comparison of stability regions using Equation (1.31) for a (a) 40%, (b) 50%, and (c) 60% duty cycle square wave vs. using Equation (1.32) for a (d) 40%, (e) 50%, and (f) 60% duty cycle square wave. Radial stability is in red and axial stability is in blue.

For a sine wave the three physical characteristics are the DC offset, amplitude, frequency. Because high voltage sine waves are generated with a circuit tuned to a single frequency, only the DC offset and amplitude are changed during a single experiment. Thus, the standard Mathieu parameters are useful because a exclusively depends on the DC offset and q exclusively depends on the amplitude. High voltage square waves are generated by switching between two power supplies, so the DC offset and amplitude are not changed during an experiment. The switch can be programmed to change duty cycle and frequency during an experiment, so a and q for a square wave should individually represent these two physical characteristics. One possible definition is to simply define a as the duty cycle and use the definition of q from Equation (1.32). Then, a change in duty cycle will only change a and a change in frequency will only change q. Figure 1.4 shows the stability region using this definition.

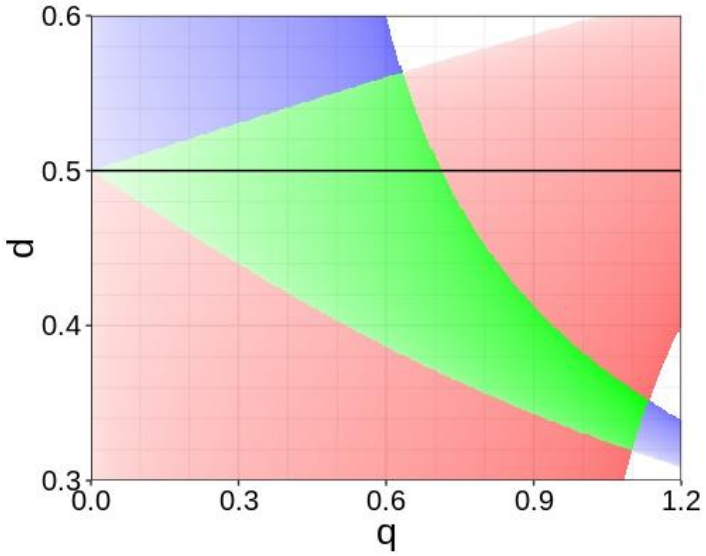

Figure 1.4. Stability diagram for a square wave 3D ion trap using duty cycle as the y-axis and the definition for q from Equation (1.32) for the x-axis. Radial stability is in red and axial stability is in blue.

1.2.4 The Pseudopotential Well Depth

To describe stability more completely in an ion trap, ion energy must also be considered. Calculating the maximum energy an ion can have based on its frequency provides a simple model that predicts stability based on ion energy called the pseudopotential well depth or Dehmelt approximation for well depth [14]. Starting with Equation (1.1) and using the ion's secular frequency to describe its acceleration gives:

$$\frac{d^2u}{dt^2} = -\omega_u^2 u = \frac{ze}{m}\frac{dD_u}{du} \tag{1.33}$$

where ω_u is the secular frequency and u is the ion displacement along the u dimension. The variable D_u is used in place of V to describe well depth potential due to secular ion motion rather than an actual applied voltage. Typically, a stability parameter defined as $\beta = \frac{2\omega}{\Omega}$ is used as a unitless definition of ion secular frequency, where any stable ion has $0 \leq \beta \leq 1$. Substitution of this parameter into Equation (1.33) and rearrangement gives:

$$\frac{dD_u}{du} = -\frac{m\beta_u^2\Omega^2 u}{4ze} \tag{1.34}$$

After integration, Equation (1.34) becomes:

$$D_u = -\frac{m\beta_u^2\Omega^2 u_0^2}{8ze} \tag{1.35}$$

Substitution of the q parameters from Equations (1.9) and (1.10) yields:

$$D_u = -\frac{\beta_u^2 V}{4q_u} \tag{1.36}$$

Because β is not readily calculated for any type of waveform, there exist competing approaches to determine exact well depth potentials [15–17]. For $q < 0.4$ in a sine wave ion trap, $\beta \approx \sqrt{a + \frac{q^2}{2}}$. If $a = 0$, Equation (1.36) simplifies to:

$$D_u \approx -\frac{q_u V}{8} \tag{1.37}$$

For a square wave trap, β is determined from the transfer matrix M (Equation (1.30)) as $\beta = \frac{1}{\pi}\text{acos}\frac{M_{11}+M_{22}}{2}$. A useful approximation can be used for $q < 0.3$ [18]:

$$D_u \approx -0.206 q_u V \tag{1.38}$$

The well depth potential predicts the maximum energy per charge an ion can have without leaving the ion trap. To express well depth potential in terms of ion energy rather than a voltage potential, Equations (1.37) and (1.38) can be multiplied by ion charge. Theory and experiment suggest that ions are thermalized to roughly room temperature after experiencing collisions with a neutral background gas in an ion trap [19,20]. Thus, the minimum well depth potential energy, or $z \times D_u$, is $-\frac{3}{2}kT \approx -0.04$ eV. Substituting this condition into Equations (1.37) and (1.38) and using the radial dimension for q gives:

$$0.04 = Czq_rV = \frac{4Cz^2eV^2}{m\Omega^2(r_0^2 + 2x_0^2)} \tag{1.39}$$

where $C = 0.125$ for a sine wave and 0.206 for a square wave. Solving for ion mass and charge gives:

$$\frac{m}{z^2} = \frac{100CeV^2}{\Omega^2(r_0^2 + 2x_0^2)} \tag{1.40}$$

Whereas the low mass cut off is determined by ion mass-to-charge, this approach suggests that the high mass cut off of an ion trap is determined by ion mass divided by charge squared. In practice, this means that under identical trapping conditions, a highly charged ion of some *m/z* will be trapped more stably than another ion with a similar *m/z* but less overall charge.

1.3 The 3D Ion Trap as a Reaction Cell

Because the 3D ion trap can stably trap ions for long periods of time, many types of reactions including photoreactions [5,21] ion/molecule [4,22,23], and ion/ion reactions [24–26]. Such reactions can produce ion fragments used in identification and characterization [11,27] and simplify spectra of multiply-charged ions [28,29]. These reactions coupled with ion isolation [30–32] from the basis of tandem MS (MS^n). As a tandem-in-time mass spectrometer (i.e., steps in a MS^n experiment occur in the same space, namely the ion trap), the ion trap can perform many steps of MS efficiently [33].

1.3.1 Ion Isolation

The stability diagrams in Figure 1.2 and Figure 1.4 illustrate how a narrow range of *m/z* can be isolated using the boundaries of stability. When using a sine wave, the *m/z* of interest is moved under the apex of the stability region by adjusting the sine wave amplitude (i.e., changing the ion's *q* value to 0.781), and then adjust the DC offset to move the *m/z* of interest into the apex (i.e., changing the ion's *a* value to 0.150) [30]. Because there is a practical limit to the sine wave amplitude, this "apex isolation" only works for lower *m/z* ions. Broadband isolation techniques that eject large ranges of *m/z* via excitation can replace ejection using stability boundaries to isolate higher *m/z* ions [34,35]. When using a frequency-flexible square wave; however, any *m/z* can be moved to high *q* values because lower frequencies (and therefore lower powers) correspond to higher *q* values. In this case apex isolation is effected by moving the *m/z* of interest below the

apex by changing the square wave frequency and then changing the duty cycle to move the ion into the apex [32]. The apex corresponds to a q value of ~0.63 (using the definition of q from Equation (1.32)) and a duty cycle of ~56% [36]. Isolation via broadband excitation can be performed in a square wave ion trap as well [37].

1.3.2 Ion/Ion Reactions

Because the 3D ion trap dynamically traps in all three spatial dimensions, ions of both polarities can be stored together with enough overlap to facilitate reactions between ion populations of different polarities [7]. Ion/ion reactions can lead to a transfer of protons [38], electrons [26], or ions [39,40], through a short-lived interaction between the cation and anion. An ion/ion reaction can also yield a long-lived complex formed by the cation and anion [41–44]. Charge reduction ion/ion reactions, wherein a reagent ion is neutralized by taking excess charge from an analyte ion of interest, is a useful "chemical deconvolution" technique [28,45]. Mixtures of multiply-charge analyte ions have a large degree of overlap in *m/z*, because the difference in *m/z* for higher charge states is less than the difference in *m/z* for lower charge states. Reducing the charge on the different analyte ions generates better separated charge states for each individual analyte and better separated charge states for analytes of different mass.

1.4 The 3D Ion Trap as a Mass-to-Charge Analyzer

Ion traps typically perform mass analysis by sequentially ejecting ions according to their *m/z* [8,9]. To increase the range of *m/z* available for mass analysis, ion traps are operated along the $a = 0$ line. For a sine wave operated trap, the DC offset is set to 0; for a square wave operated trap, the duty cycle is set to 50% and the two voltages are equal and opposite (e.g., ±500 V). Equation (1.9) suggests different ways to generate a mass spectrum by changing the trapping waveform amplitude or frequency. Changing these parameters over time generates a raw time spectrum that can be correlated back to *m/z*.

1.4.1 Mass-to-Charge Instability Scan via Boundary Ejection

To create a mass spectrum with the 3D ion trap as the mass analyzer, ions must be ejected sequentially according to their *m/z*. A simple method is to change the *a* or *q* values of the trapped

ions over time. In a sine wave ion trap that has a fixed frequency and changes the amplitude, the q values of the ions can be changed by ramping the amplitude as suggested by Equation (1.9). Solving the equation for *m/z* and using the boundary condition of (a_{eject}, q_{eject}) = (0, 0.908) gives:

$$m/z = -\frac{8eV}{q_{eject}\Omega^2(r_0^2 + 2x_0^2)} \tag{1.41}$$

which calculates the *m/z* range of the boundary ejection scan based on the voltage ramp used. As a particular *m/z* passes the boundary condition, it loses stability in the axial dimension while maintaining stability in the radial dimension (see Figure 1.2). This results in ions being ejected through holes in the end cap electrodes and being detected outside the trap. However, the upper *m/z* limit is dictated by the achievable voltage range. The stability diagram also suggests that ions can be sequentially destabilized in the axial dimension by decreasing the ions' a values [46] as suggested by:

$$m/z = \frac{16eU}{a_{eject}\Omega^2(r_0^2 + 2x_0^2)} \tag{1.42}$$

This approach can measure ions that are not measurable using boundary ejection via an amplitude scan. The main barrier to this approach is that a_{eject} is dependent on *m/z*, so it creates a mass spectrum that requires a nonlinear calibration.

Square wave, or digital, operation of an ion trap provides another type of boundary ejection scan. Because the amplitude of the square wave is held constant while the frequency is changes over time, ions can be ejected through the (a_{eject}, q_{eject}) = (0, 0.712) ejection point by decreasing the square wave frequency over time [18]. The main difference of this frequency scan from the sine wave voltage scan is that *m/z* does not vary linearly with frequency. However, direct digital synthesis (DDS) technology can easily generate a frequency scan that varies according to the square of its inverse (per Equation (1.41)) to create a linear mass spectrum [47,48].

1.4.2 Mass-to-Charge Excitation Scan via Resonance Ejection

While boundary ejection provides the methodology to measure ion *m/z* after storing ions in the ion trap, the mass spectral resolution is poor. One way to improve resolution is to excite ions such that they exit the trap more quickly to reduce the time-of-flight spread in ions as they leave the trap and approach the detector. This approach is generally called axial modulation or resonance ejection, because an auxiliary dipolar waveform is applied to the end cap electrodes (in

the axial dimension) that ejects ions with secular frequencies that resonate with its frequency [49,50]. The secular frequency of ion motion is calculated from:

$$\omega = \frac{\beta\Omega}{2} \tag{1.43}$$

where ω is the ion secular frequency and β is a stability parameter approximated by [8]:

$$\beta \approx \sqrt{a + \frac{q^2}{2}} \tag{1.44}$$

for a sine wave ion trap when $q < 0.4$ and calculated by [18]:

$$\beta = \frac{1}{\pi}\operatorname{acos}\left(\frac{M_{11} + M_{22}}{2}\right) \tag{1.45}$$

for a square wave ion trap using the trace of the transfer matrix described by Equation (1.30). Standard resonance ejection ramps the q values of ions, in the same way as boundary ejection, but with a resonance ejection point defined by the auxiliary dipolar frequency. However, a *m/z* scan can also be effected by ramping the resonance ejection frequency over time while holding the trapping amplitude and frequency constant [51,52]. This in effect maintains ions at the same q values while scanning the resonance ejection point.

Another benefit of resonance ejection is the ability to measure high *m/z* ions that cannot be measured using boundary ejection [53]. In boundary ejection the maximum *m/z* is found by using the maximum voltage or minimum frequency in Equation (1.41). Resonance ejection decreases the value of q_{eject} (by decreasing β_{eject}) so that the same maximum voltage or minimum frequency corresponds to a higher *m/z*. This also increases the minimum measurable *m/z* for the same voltage or frequency ramp, so the correct resonance ejection point must be chosen with both the minimum and maximum desired *m/z* in mind.

In a sine wave ion trap, resonance ejection of higher *m/z* ions occurs at lower frequencies. An additional nuance for square wave ion traps is that over the course of a frequency scan, higher *m/z* ions are ejected at lower trapping frequencies. To eject ions at a constant ejection point, the trapping frequency and resonance ejection frequency are ramped down together [37]. This is because β is constant at the point of ejection, and therefore ω decreases as the frequency of the trapping waveform decreases. Spectral peak resolution increases as the number of cycles an ion experiences during ejection increases [54,55]; therefore ions ejected at lower frequencies could generate wider peaks.

1.5 Conclusions

The 3D QIT is a flexible tool in mass spectrometry that can act as a reaction cell and mass analyzer for tandem-in-time MS experiments. It particularly has the strength for housing ion/ion reactions and measuring reaction products through a variety of operation modes. Constant frequency sine wave operation facilitates larger *m/z* trapping ranges benefitting the reaction cell characteristics, and constant amplitude square wave operation facilitates larger and more accessible *m/z* scanning ranges benefitting the mass analyzer characteristics.

1.6 References

[1] E.N. Nikolaev, Y.I. Kostyukevich, G.N. Vladimirov, Fourier transform ion cyclotron resonance (FT ICR) mass spectrometry: Theory and simulations, Mass Spectrom. Rev. 35 (2016) 219–258. https://doi.org/10.1002/mas.21422.

[2] Q. Hu, R.J. Noll, H. Li, A. Makarov, M. Hardman, R.G. Cooks, The Orbitrap: A new mass spectrometer, J. Mass Spectrom. 40 (2005) 430–443. https://doi.org/10.1002/jms.856.

[3] I. V. Chernushevich, A. V. Loboda, B.A. Thomson, An introduction to quadrupole-time-of-flight mass spectrometry, J. Mass Spectrom. 36 (2001) 849–865. https://doi.org/10.1002/jms.207.

[4] F. Vedel, M. Vedel, J.S. Brodbelt, Ion/Molecule Reactions, in: R.E. March, J.F.J. Todd (Eds.), Pract. Asp. Ion Trap Mass Spectrom. Vol. 1, CRC Press, Boca Raton, 1995.

[5] J.S. Brodbelt, J.J. Wilson, Infrared multiphoton dissociation in quadrupole ion traps, Mass Spectrom. Rev. 28 (2009) 390–424. https://doi.org/10.1002/mas.20216.

[6] J.N. Louris, J.S. Brodbelt, R.G. Cooks, Photodissociation in a quadrupole ion trap mass spectrometer using a fiber optic interface, Int. J. Mass Spectrom. Ion Process. 75 (1987) 345–352. https://doi.org/10.1016/0168-1176(87)83045-8.

[7] S.A. McLuckey, J.L. Stephenson, Ion/ion chemistry of high-mass multiply charged ions, Mass Spectrom. Rev. 17 (1998) 369–407. https://doi.org/10.1002/(SICI)1098-2787(1998)17:6<369::AID-MAS1>3.0.CO;2-J.

[8] R.E. March, An introduction to quadrupole ion trap mass spectrometry, J. Mass Spectrom. 32 (1997) 351–369. https://doi.org/10.1002/(SICI)1096-9888(199704)32:4<351::AID-JMS512>3.0.CO;2-Y.

[9] R.E. March, J.F. Todd, Quadrupole ion trap mass spectrometry, 2nd edition, Wiley-Interscience, Hoboken, NJ, 2005.

[10] Y. Xia, P.A. Chrisman, D.E. Erickson, J. Liu, X. Liang, F.A. Londry, M.J. Yang, S.A. McLuckey, Implementation of ion/ion reactions in a quadrupole/time-of-flight tandem mass spectrometer, Anal. Chem. 78 (2006) 4146–4154. https://doi.org/10.1021/ac0606296.

[11] T.-Y. Kim, M.S. Thompson, J.P. Reilly, Peptide photodissociation at 157 nm in a linear ion trap mass spectrometer, Rapid Commun. Mass Spectrom. 19 (2005) 1657–1665. https://doi.org/10.1002/rcm.1969.

[12] N. V. Konenkov, M. Sudakov, D.J. Douglas, Matrix methods for the calculation of stability diagrams in quadrupole mass spectrometry, J. Am. Soc. Mass Spectrom. 13 (2002) 597–613. https://doi.org/10.1016/S1044-0305(02)00365-3.

[13] M. Sudakov, E. Nikolaev, Ion Motion Stability Diagram for Distorted Square Waveform Trapping Voltage, Eur. J. Mass Spectrom. 8 (2002) 191–199. https://doi.org/10.1255/ejms.491.

[14] H.G. Dehmelt, Radiofrequency spectroscopy of stored ions I: Storage, in: D.R. Bates (Ed.), Adv. At. Mol. Physics, Vol. 3, Academic, New York, 1967: pp. 53–72.

[15] D.J. Douglas, A.S. Berdnikov, N. V. Konenkov, The effective potential for ion motion in a radio frequency quadrupole field revisited, Int. J. Mass Spectrom. 377 (2015) 345–354. https://doi.org/10.1016/j.ijms.2014.08.009.

[16] P.T.A. Reilly, G.F. Brabeck, Mapping the pseudopotential well for all values of the Mathieu parameter q in digital and sinusoidal ion traps, Int. J. Mass Spectrom. 392 (2015) 86–90. https://doi.org/10.1016/j.ijms.2015.09.013.

[17] A.S. Berdnikov, D.J. Douglas, N. V. Konenkov, The pseudopotential for quadrupole fields up to q = 0.9080, Int. J. Mass Spectrom. 421 (2017) 204–223. https://doi.org/10.1016/j.ijms.2017.04.003.

[18] F.L. Brancia, L. Ding, Rectangular waveform driven digital ion trap (DIT) mass spectrometer: Theory and applications, in: R.E. March, J.F.J. Todd (Eds.), Pract. Asp. Trapped Ion Mass Spectrom. Vol. IV Theory Instrum., CRC Press, Boca Raton, 2010.

[19] D.E. Goeringer, S.A. Mcluckey, Evolution of ion internal energy during collisional excitation in the Paul ion trap: A stochastic approach ARTICLES YOU MAY BE INTERESTED IN Time-of-Flight Mass Spectrometer with Improved Resolution, J. Chem. Phys. 104 (1996) 2214. https://doi.org/10.1063/1.471812.

[20] S. Gronert, Estimation of effective ion temperatures in a quadrupole ion trap, J. Am. Soc. Mass Spectrom. 9 (1998) 845–848. https://doi.org/10.1016/S1044-0305(98)00055-5.

[21] J.S. Brodbelt, Photodissociation mass spectrometry: New tools for characterization of biological molecules, Chem. Soc. Rev. 43 (2014) 2757–2783. https://doi.org/10.1039/c3cs60444f.

[22] B.M. Prentice, S.A. Mcluckey, Dipolar DC Collisional Activation in a “Stretched” 3-D Ion Trap: The Effect of Higher Order Fields on rf-Heating, J. Am. Soc. Mass Spectrom. 23 (2012) 736–744. https://doi.org/10.1007/s13361-011-0303-9.

[23] F. Xu, L. Wang, X. Dai, X. Fang, C.-F. Ding, Resonance Activation and Collision-Induced-Dissociation of Ions Using Rectangular Wave Dipolar Potentials in a Digital Ion Trap Mass Spectrometer, J. Am. Soc. Mass Spectrom. 25 (2014) 556–562. https://doi.org/10.1007/s13361-013-0804-9.

[24] B.M. Prentice, S.A. McLuckey, Gas-phase ion/ion reactions of peptides and proteins: Acid/base, redox, and covalent chemistries, Chem. Commun. 49 (2013) 947–965. https://doi.org/10.1039/c2cc36577d.

[25] M. He, J.F. Emory, S.A. McLuckey, Reagent Anions for Charge Inversion of Polypeptide/Protein Cations in the Gas Phase, J. Am. Soc. Mass Spectrom. 63 (1991) 3173–3182. https://doi.org/10.1021/ac0482312.

[26] S.J. Pitteri, P.A. Chrisman, J.M. Hogan, S.A. McLuckey, Electron transfer ion/ion reactions in a three-dimensional quadrupole ion trap: Reactions of doubly and triply protonated peptides with SO 2•-, Anal. Chem. 77 (2005) 1831–1839. https://doi.org/10.1021/ac0483872.

[27] D.F. Hunt, J.R. Yates, J. Shabanowitz, S. Winston, C.R. Hauer, Protein sequencing by tandem mass spectrometry, Proc. Natl. Acad. Sci. U. S. A. 83 (1986) 6233–6237. https://doi.org/10.1073/pnas.83.17.6233.

[28] J.L. Stephenson, S.A. McLuckey, Simplification of Product Ion Spectra Derived from Multiply Charged Parent Ions via Ion/Ion Chemistry, Anal. Chem. 70 (1998) 3533–3544. https://doi.org/10.1021/ac9802832.

[29] K.J. Laszlo, M.F. Bush, Analysis of Native-Like Proteins and Protein Complexes Using Cation to Anion Proton Transfer Reactions (CAPTR), J. Am. Soc. Mass Spectrom. 26 (2015) 2152–2161. https://doi.org/10.1007/s13361-015-1245-4.

[30] J.N. Louris, J.S. Brodbelt-Lustig, R. Graham Cooks, G.L. Glish, G.J. van Berkel, S.A. McLuckey, Ion isolation and sequential stages of mass spectrometry in a quadrupole ion trap mass spectrometer, Int. J. Mass Spectrom. Ion Process. 96 (1990) 117–137. https://doi.org/10.1016/0168-1176(90)87025-C.

[31] S.A. McLuckey, D.E. Goeringer, G.L. Glish, Selective ion isolation/rejection over a broad mass range in the quadrupole ion trap, J. Am. Soc. Mass Spectrom. 2 (1991) 11–21. https://doi.org/10.1016/1044-0305(91)80056-D.

[32] F.L. Brancia, B. McCullough, A. Entwistle, J.G. Grossmann, L. Ding, Digital asymmetric waveform isolation (DAWI) in a digital linear ion trap, J. Am. Soc. Mass Spectrom. 21 (2010) 1530–1533. https://doi.org/10.1016/j.jasms.2010.05.003.

[33] J. V Johnson, R.A. Yost, P.E. Kelley, D.C. Bradford, Tandem-in-Space and Tandem-in-Time Mass Spectrometry: Triple Quadrupoles and Quadrupole Ion Traps, Anal. Chem. 62 (1990) 2162–2172. https://doi.org/10.1021/ac00219a003.

[34] S. Guan, A.G. Marshall, Stored waveform inverse Fourier transform (SWIFT) ion excitation in trapped-ion mass spectometry: Theory and applications, Int. J. Mass Spectrom. Ion Process. 157–158 (1996) 5–37. https://doi.org/10.1016/S0168-1176(96)04461-8.

[35] M.H. Son, R. Graham Cooks, Selective Injection and Isolation of Ions in Quadrupole Ion Trap Mass Spectrometry Using Notched Waveforms Created Using the Inverse Fourier Transform, Wiley, 1994. https://pubs.acs.org/sharingguidelines (accessed September 4, 2020).

[36] G.F. Brabeck, P.T.A. Reilly, Mapping ion stability in digitally driven ion traps and guides, Int. J. Mass Spectrom. 364 (2014) 1–8. https://doi.org/10.1016/j.ijms.2014.03.008.

[37] L. Ding, M. Sudakov, F.L. Brancia, R. Giles, S. Kumashiro, A digital ion trap mass spectrometer coupled with atmospheric pressure ion sources, J. Mass Spectrom. 39 (2004) 471–484. https://doi.org/10.1002/jms.637.

[38] J.L. Stephenson, S.A. McLuckey, Ion/ion proton transfer reactions for protein mixture analysis, Anal. Chem. 68 (1996) 4026–4032. https://doi.org/10.1021/ac9605657.

[39] K.A. Newton, M. He, R. Amunugama, S.A. McLuckey, Selective cation removal from gaseous polypeptide ions: Proton vs. sodium ion abstraction via ion/ion reactions, Phys. Chem. Chem. Phys. 6 (2004) 2710–2717. https://doi.org/10.1039/b315240e.

[40] H.P. Gunawardena, R.A.J. O'hair, S.A. Mcluckey, Selective Disulfide Bond Cleavage in Gold(I) Cationized Polypeptide Ions Formed via Gas-Phase Ion/Ion Cation Switching, (2006). https://doi.org/10.1021/pr0602794.

[41] J. Wu, S.A. McLuckey, Ion/ion reactions of multiply charged nucleic acid anions: Electron transfer, proton transfer, and ion attachment, Int. J. Mass Spectrom. 228 (2003) 577–597. https://doi.org/10.1016/S1387-3806(03)00165-9.

[42] M. He, S.A. McLuckey, Two ion/ion charge inversion steps to form a doubly protonated peptide from a singly protonated peptide in the gas phase, J. Am. Chem. Soc. 125 (2003) 7756–7757. https://doi.org/10.1021/ja0354521.

[43] H.-C. Chao, M. Shih, S.A. Mcluckey, Generation of Multiply Charged Protein Anions from Multiply Charged Protein Cations via Gas-Phase Ion/Ion Reactions, J. Am. Soc. Mass Spectrom. 31 (2020) 56. https://doi.org/10.1021/jasms.0c00062.

[44] H.P. Gunawardena, S.A. McLuckey, Synthesis of multi-unit protein hetero-complexes in the gas phase via ion–ion chemistry, J. Mass Spectrom. 39 (2004) 630–638. https://doi.org/10.1002/jms.629.

[45] J. Stephenson, S.A. McLuckey, Ion/Ion reactions for oligopeptide mixture analysis: Application to mixtures comprised of 0.5-100 kDa components, J. Am. Soc. Mass Spectrom. 9 (1998) 585–596. https://doi.org/10.1016/S1044-0305(98)00025-7.

[46] B.M. Prentice, S.A. Mcluckey, Analysis of High Mass-to-Charge Ions in a Quadrupole Ion Trap Mass Spectrometer via an End-Cap Quadrupolar Direct Current Downscan, Anal. Chem. 84 (2012) 26. https://doi.org/10.1021/ac301741a.

[47] K.W. Lee, G.S. Eakins, M.S. Carlsen, S.A. McLuckey, Increasing the Upper Mass/Charge Limit of a Quadrupole Ion Trap for Ion/Ion Reaction Product Analysis via Waveform Switching, J. Am. Soc. Mass Spectrom. 30 (2019) 1126–1132. https://doi.org/10.1007/s13361-019-02156-z.

[48] K.W. Lee, G.S. Eakins, M.S. Carlsen, S.A. McLuckey, Ion trap operational modes for ion/ion reactions yielding high mass-to-charge product ions, Int. J. Mass Spectrom. 451 (2020) 116313. https://doi.org/10.1016/j.ijms.2020.116313.

[49] R.E. Kaiser, J.N. Louris, J.W. Amy, R.G. Cooks, D.F. Hunt, Extending the mass range of the quadrupole ion trap using axial modulation, Rapid Commun. Mass Spectrom. 3 (1989) 225–229. https://doi.org/10.1002/rcm.1290030706.

[50] J.D. Williams, K.A. Cox, R.G. Cooks, S.A. McLuckey, K.J. Hart, D.E. Goeringer, Resonance Ejection Ion Trap Mass Spectrometry and Nonlinear Field Contributions: The Effect of Scan Direction on Mass Resolution, Anal. Chem. 66 (1994) 725–729. https://doi.org/10.1021/ac00077a023.

[51] D.T. Snyder, C.J. Pulliam, J.S. Wiley, J. Duncan, R.G. Cooks, Experimental Characterization of Secular Frequency Scanning in Ion Trap Mass Spectrometers, 27 (2016) 1243–1255. https://doi.org/10.1007/s13361-016-1377-1.

[52] D.T. Snyder, C.J. Pulliam, R.G. Cooks, Linear mass scans in quadrupole ion traps using the inverse Mathieu q scan, Rapid Commun. Mass Spectrom. (2016) 2369–2378. https://doi.org/10.1002/rcm.7710.

[53] R.E. Kaiser, R. Graham Cooks, G.C. Stafford, J.E.P. Syka, P.H. Hemberger, Operation of a quadrupole ion trap mass spectrometer to achieve high mass/charge ratios, Int. J. Mass Spectrom. Ion Process. 106 (1991) 79–115. https://doi.org/10.1016/0168-1176(91)85013-C.

[54] E. Fischer, Die dreidimensionale Stabilisierung von Ladungsträgern in einem Vierpolfeld, Zeitschrift Für Phys. 156 (1959) 1–26. https://doi.org/10.1007/BF01332512.

[55] J.C. Schwartz, J.E.P. Syka, I. Jardine, High resolution on a quadrupole ion trap mass spectrometer, J. Am. Soc. Mass Spectrom. 2 (1991) 198–204. https://doi.org/10.1016/1044-0305(91)80044-8.

CHAPTER 2. INCREASING THE UPPER MASS/CHARGE LIMIT OF A QUADRUPOLE ION TRAP FOR ION/ION REACTION PRODUCT ANALYSIS VIA WAVEFORM SWITCHING

2.1 Introduction

Quadrupole ion traps (QITs) [1], both 3-D and linear, are widely used in mass spectrometry (MS) in part due to their small size and versatility in executing multi-stage mass spectrometry experiments (i.e., MS^n), albeit with moderate-to-low resolution at usual scan rates [2]. Ion traps are particularly useful as vessels for ionic reactions due to their ability to trap ions of both polarities simultaneously over relatively wide ranges of mass-to-charge (*m/z*). This capability has enabled the development of tandem-in-time experiments [3] involving, for example, collisional activation [4], photodissociation [5–7], and ion/molecule reactions [8]. The coupling of electrospray ionization (ESI) [9,10], with its propensity for generating multiply-charged ions from relatively large molecules, with ion traps [11] has enabled the study of the reactions of oppositely-charged ions [12–14]. Ion/ion reactions, *inter alia*, have proven to be robust means for charge state manipulation of high mass ions and have been used to facilitate protein mixture analysis [15], concentration of multiple charge-states into a single lower charge state [16], product ion mass determination following a dissociation reaction [17], and inversion of ions from one polarity to another [18].

Electrospray ionization, due to the multiple-charging effect, tends to generate ions over a relatively narrow *m/z* range. For proteins under denaturing conditions, for example, it is common to observe charge states over a range of *m/z* 500–2000 [19]. Under 'native MS' conditions, protein and protein complex ions can exceed *m/z* 10,000 [20], although for a given protein or protein complex the charge state distribution tends to be narrow. In any case, ion/ion proton transfer reactions convert lower *m/z* ions to higher *m/z* ions. In the case of MS^n experiments, it is desirable to generate and transfer ions over a relatively narrow *m/z* range, thereby minimizing mass discrimination effects in the interface, ion transport devices, and the capturing of ions injected into

the trap. It is of interest to maximize MS^{n-1} performance as well as to optimize the final MS step after ions have been reduced in charge to give high *m/z* ions. Expanding the *m/z* range over which the QIT can be used for mass analysis is desirable for the application of ion/ion reactions to large proteins and protein complexes. The main factors that can limit the upper *m/z* in an ion trap ion/ion reaction experiment using mass-selective instability [21] for mass analysis are: 1) external detector response, 2) the difference in *m/z* ratios of the reagent ions and product ions as well as the range over which both can be stored simultaneously, and 3) the *m/z* range associated with the mass analysis step. In this work, we describe a novel approach to maintaining MS^{n-1} performance while improving upon the final MS step by switching from a high frequency sine wave waveform used for ion accumulation and ion/ion reaction to a low-frequency square wave for mass analysis in order to address limitations associated with the third potential limitation mentioned above.

2.2 Experimental

2.2.1 Materials

Perfluoromethyldecalin (PMD, 512 Da) was purchased from Oakwood Chemicals (Estill, SC). Acetic acid was purchased from Avantor (Radnor, PA). LC/MS grade water was purchased from Fisher Scientific (Hampton, NH). Bovine serum albumin (BSA, 66.4 kDa) and Human immunoglobulin G (IgG, 150 kDa) were purchased from MilliporeSigma (St. Louis, MO). Solutions (1 mg/mL) of denatured BSA and IgG were prepared by dissolving them in 99:1 (v/v) water / acetic acid. PMD was placed in a small glass vial, and vapors from the open vial were sampled into the atmospheric sampling glow discharge ionization (ASGDI) source [22].

2.2.2 Mass Spectrometry

This work demonstrates a method to analyze high *m/z* ions in a QIT that are initially present at relatively low *m/z* and are transformed to lower *z* via ion/ion reactions. Nano-electrospray ionization (nESI) was used to introduce highly charged protein cations into the QIT. Subsequent introduction of and mutual storage with oppositely charged reagent ions reduces the charges of the protein ions. Switching the drive RF from a high frequency sine wave to a low frequency square wave simultaneously ejects the reagent anions and creates better trapping conditions for the high

m/z product ions. A frequency scan of the square wave ejects the product ions according to *m/z* for mass analysis.

To operate the QIT with two different types of waveforms, custom software was developed, and an instrument controller was made. Figure 2.1 is a schematic of the instrumental setup. All functions were performed by a Hitex Shieldbuddy TC275 (Hitex UK Ltd.) development board. The Shieldbuddy was programmed with the Arduino IDE to understand instructions given to it from a custom Python console program. The instrument controller provided TTL outputs to switch voltages on lenses such that either positive ions from the ESI source or negative ions from the ASGDI source would be transported to the QIT and to trigger detection events. The controller also provided a −10 to 10 V analog output as a reference value for a custom-built sine wave generator. A clock input of 1.008 MHz dictated the frequency of the sine wave. The generated sine wave was amplified with a tank circuit and connected to the ring electrode of the QIT. Within the controller circuit board, the Shieldbuddy was connected to a direct digital synthesizer (DDS) circuit. The output of the DDS was used as a trigger to a custom-built high voltage pulse generator that was supplied with +200 and −200 V. The output of the pulse generator was connected to both end cap electrodes of the QIT. Samples of denatured BSA and IgG were introduced to the instrument with nESI using +1000 to +1500 V applied to a wire inserted into the back of the emitter capillary [23]. Perfluoromethyldecalin (PMD) vapor was introduced through tubing into the glow discharge region. Typically −400 to −500 V was applied to the front plate to create the discharge.

Figure 2.2 illustrates a typical scan function. The protein was first introduced to the QIT followed by PMD anions while the sine wave was applied to the ring electrode. After a determined amount of time for the ion/ion reaction (typically hundreds of ms), the resulting post-ion/ion reaction spectrum was obtained either via resonance ejection (sine wave) or frequency scanning (square wave). In the case of resonance ejection, a short RF-amplitude ramp from 3,000 V to 5,050 V was applied directly after the ion/ion reaction period to eject residual PMD anions. The RF-amplitude was then reduced to as low as 550 V and scanned up to roughly 5,050 V while simultaneously applying a resonance ejection signal to the end cap electrodes. In the case of the digital ion trap (DIT) frequency scan, the low frequency square wave was applied to the end cap electrodes while the sine wave was still on. After at least 10 ms, the sine wave was turned off. Without this overlap of the two trapping waveforms, significant ion loss was observed. The frequency was scanned down (according to the inverse of the frequency squared) to eject product

ions in increasing *m/z* order, linearly. Ions leaving the trap were accelerated to a detector consisting of a conversion dynode and electron multiplier. Because the low frequency waveforms applied to the end caps interfered with the detected signal, the signal was filtered with a low pass RC circuit before being measured. Peaks from collected time spectra were manually fit according to the range of frequencies used and relative peak spacings to determine their *m/z* values, and the time spectra were converted to mass spectra.

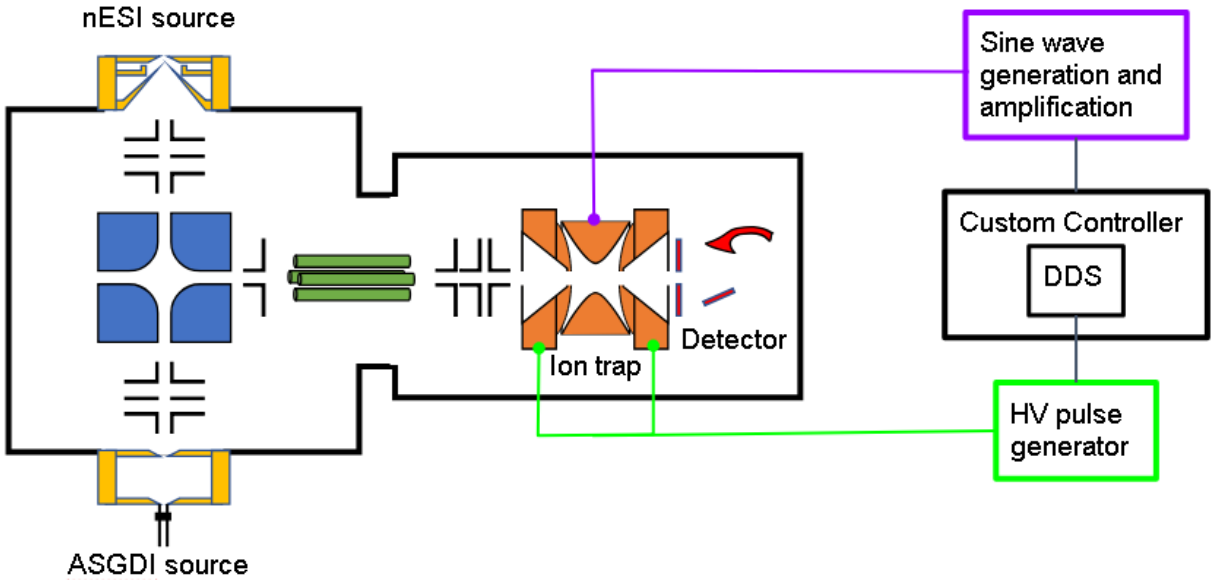

Figure 2.1. Schematic of instrumental setup for waveform switching experiments. A high frequency sine wave is applied to the ring electrode during ion injection and mutual storage to provide adequate trapping of low *m/z* reagent and analyte ions and high *m/z* product ions. A low frequency square wave is applied to the end cap electrodes during mass analysis to provide better confinement of very high *m/z* product ions prior to mass selective ejection.

Comparison spectra were measured using resonance ejection. Circuitry and scan functions for resonance ejection experiments are not represented. In the case of the resonance ejection scan using low frequencies, the sampling rate of the detection electronics was sufficiently fast to observe individual ion packets that appear as equally spaced peaks with a period corresponding to the ejection frequency.

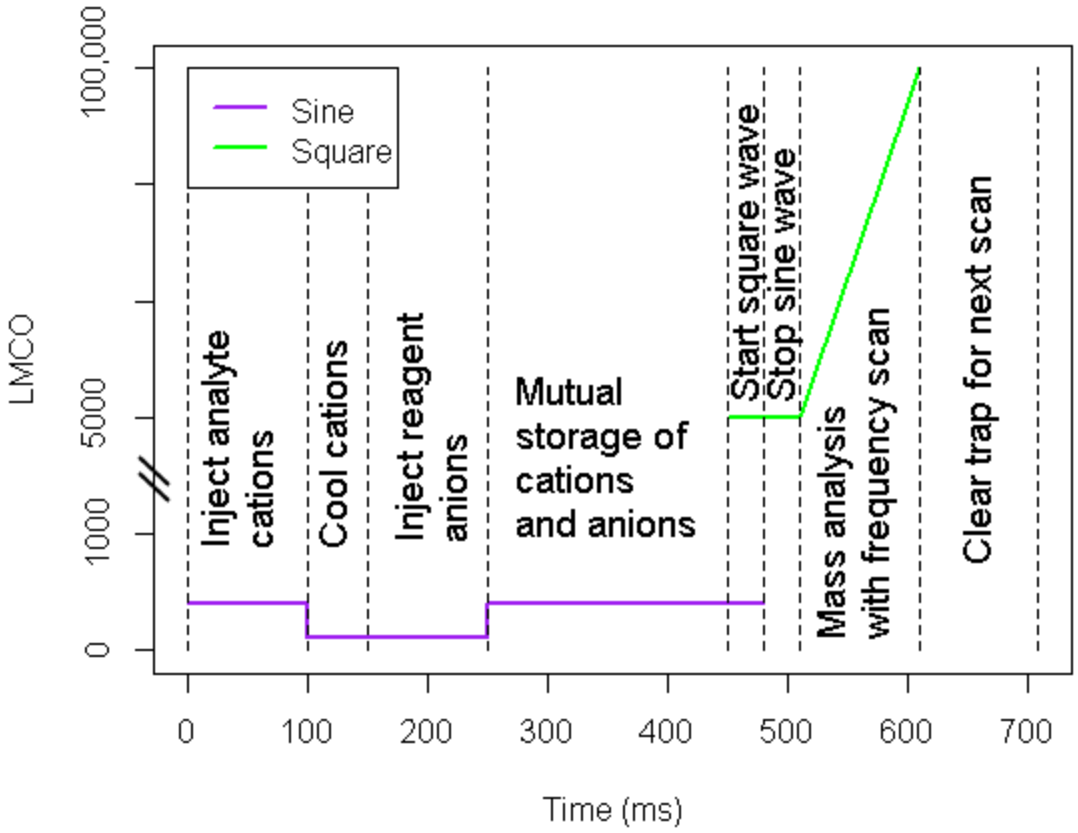

Figure 2.2. Typical scan function for waveform switching experiment. A high frequency sine wave traps analyte and reagent ions during injection and mutual storage. At the end of the mutual storage step, a low frequency square wave is applied to provide better trapping of high *m/z* product ions. The sine wave is then turned off, and mass analysis is accomplished with a frequency scan of the square wave.

2.3 Results and Discussion

A traditional sine wave driven QIT stores ions that simultaneously fall within stability boundaries in the radial and axial dimensions and in regions where the pseudopotential well depth [24] is sufficiently deep to avoid ion evaporation from the trap. The dimensionless Mathieu parameters for sine wave 3-D ion traps are given as [1]:

$$a_z = \frac{8eU}{(m/z)\Omega^2 r_0^2} \qquad a_r = \frac{-4eU}{(m/z)\Omega^2 r_0^2} \tag{2.1}$$

$$q_z = \frac{-4eV}{(m/z)\Omega^2 r_0^2} \qquad q_r = \frac{2eV}{(m/z)\Omega^2 r_0^2} \tag{2.2}$$

where U and V are the DC offset and AC amplitude of the applied waveform, respectively, Ω is the frequency of the waveform, r_0 is the radius of the trap, and m/z is the mass-to-charge ratio of the ion of interest. Stable solutions are typically represented on a plot of 'a vs. q'. This stability

diagram predicts if an ion of a particular *m/z* will have stable periodic motion in a trap with a given radius and drive frequency. Most QITs are operated by holding 'a' equal to zero at all times (i.e., the DC component of the quadrupolar field is 0 V). A useful parameter for estimating the effective trapping potential is the so-called potential well depth approximation, D_u, which, for a sine wave driven ion trap is given by [25,26]:

$$D_u = 0.125 q_u V \tag{2.3}$$

where u represents r or z and when the condition $q_u < 0.4$ is satisfied. In the absence of a quadrupolar DC voltage, the upper *m/z* limit for ion storage in a 3-D QIT is determined by D_u. The upper *m/z* limit for mass analysis, on the other hand, might be further limited by a practical constraint, such as the accessible amplitude of the drive RF. For example, a mass-selective instability scan using boundary ejection from a sine wave QIT requires sufficient RF voltage amplitude to bring an ion to $q_z = 0.908$. This possible limitation, however, has been circumvented by ejecting ions at much lower *q*-values using resonance ejection [27]. A mass spectrum can be obtained by scanning the RF-amplitude with a fixed supplemental frequency or by scanning the supplemental frequency at a fixed RF-amplitude [28,29]. The former approach leads to ejection at a fixed *q*-value, which facilitates mass calibration and tuning of the resonance ejection voltage, whereas the latter varies the *q*-value ejection point such that the D_u-values at the ejection point also vary during the scan. An alternative approach is to scan a DC voltage applied to the ring electrode [30] or to both end cap electrodes [31], referred to as a 'down-scan', leading to a scan of *a*-values that cross the $\beta_z = 0$ stability boundary. The down-scan was shown to provide a higher achievable upper *m/z* limit than a resonance ejection scan on the same platform but with compromised resolution and a non-linear mass scale [31].

An alternate method of effecting a mass selective instability scan via boundary ejection using a QIT is to ramp the RF frequency at fixed amplitude rather than scanning the amplitude at fixed frequency [32–35]. The advantage is that it is possible to generate deeper well depths for high *m/z* ions under readily accessible voltage conditions using lower drive frequencies. Sine wave operation for a QIT is usually done at fixed frequency using a tuned circuit to minimize power consumption. Frequency scanning of a QIT, however, is readily accomplished via the switching period between two high voltage sources. Operated in this way, ion traps are often referred to as digital ion traps (DITs) [36]. In the case of a 50% duty cycle square wave DIT, the q_z-value for boundary ejection is 0.712 and the well depth at $q_u < 0.3$ is approximated by [37,38]:

$$D_u = 0.206 q_u V \quad (2.4)$$

For both sine wave and square wave operation $D_z = 2D_r$ under their respective conditions for which their respective approximations are valid. Thus, D_r is the limiting well depth value when operating at low q-values.

The objective of this work was to explore the possibility of operating a QIT using a high frequency sine wave for all stages of a multi-step experiment involving ion/ion reactions up to and including MS^{n-1} and then switching to DIT operation for the final mass analysis step. Figure 2.3a shows the post-ion/ion reaction spectrum from the reactions of a distribution of bovine serum albumin (BSA) ions of charge 45^+ to 65^+ with anions derived from glow discharge ionization of perfluoromethyldecalin (PMD) using a 2.2 kHz resonance ejection frequency (q_z-value of 0.003) combined with amplitude scanning (150 – 300 mV) at a fixed sine wave of 1.008 MHz over an RF amplitude range of 550 – 5,050 $V_{0\text{-}p}$. The RF-voltage amplitude during the ion/ion reaction was 3,000 V, which corresponds to a low mass cut-off of 370 Th and D_r = 0.9 V for BSA^+. At the end of the ion/ion reaction period, the RF amplitude was ramped to 5,050 V over a period of 30 ms to eject residual reagent anions. The insert to Figure 2.3a shows a resonance ejection scan over the RF amplitude range of 2,050 – 5,050 V with otherwise identical conditions. Peaks in the measured charge states show individual ion packets being ejected at the resonance ejection frequency due to the fast sampling rate of detection electronics. The main spectrum of Figure 2.3a shows signals corresponding to BSA^{3+} and BSA^{2+} with no evidence for BSA^+. The scanned *m/z* range corresponded to a nominal *m/z* range of 10,100 – 92,800 in 100 ms, yielding an approximate scan rate of 827 kTh/s. However, dropping the RF amplitude to 550 V to initiate the scan also reduced D_r for BSA^+ to 0.03 V, which could lead to loss of high *m/z* ions. By initiating the scan at 2,050 V, thereby maintaining a minimum D_r value of 0.4 V for BSA^+, a weak signal for BSA^+ could be observed while reducing the scanned *m/z* range to 37,700 – 92,800. We found that the BSA^+ signal disappeared after anions were ejected and RF amplitudes were decreased to less than 2,000 V prior to scanning, which suggests that D_u values of roughly 0.4 V or greater are needed to store BSA^+ ions sufficiently well to be observed upon mass analysis. We note that it has been shown that the electric field associated with the presence of a population of low *m/z* ions can assist in the storage of much higher *m/z* ions of opposite polarity, which has been termed 'trapping by proxy' [39]. This phenomenon may occur during the mutual storage ion/ion reaction period but does not occur during the mass analysis step as the anions are removed prior to scanning. We anticipate that there

is a range of D_u values over which ion storage and ejection efficiencies increases from zero to a maximum value.

Figure 2.3b shows the ion/ion product ion spectrum using the same mutual ion storage conditions for the ion/ion reaction as those for Figure 2.3a using a DIT boundary ejection frequency scan of 100 – 19 kHz at an amplitude of ±200 V. Because the switch to the low frequency square wave created a high LMCO, no extra step was needed to eject residual reagent anions prior to mass analysis. The scan covered a nominal *m/z* range of 3,200 – 88,600. This spectrum shows strong signals for both BSA^{2+} and BSA^{+}, which indicates that the mutual storage conditions were able to trap BSA^{+} ions with good efficiency. The low BSA^{+} signal associated with the insert of Figure 2.3a is therefore interpreted as arising from poor resonance ejection efficiency. The size of the trapped ion cloud is inversely related to well depth. We expect radial confinement of the ion cloud to be more important than axial confinement in determining ejection efficiency, therefore the ion cloud may be radially too large to be ejected efficiently through the exit aperture of the end cap electrode with a D_r value of 1.4 V. In the case of the frequency scan, the D_r value for BSA^{+} was 0.7 V at 100 kHz and increased to roughly 15 V at its ejection frequency of 22.06 kHz. The factor of ten increase in D_r may account for the increase in observed signal. The increased confinement prior to ejection might also account for the noticeably narrower charge states measured by the frequency scan. Therefore, the limitation associated with the BSA experiment using resonance ejection at a low q_z-value is overcome by using a DIT RF frequency scan with boundary ejection at $q_z = 0.712$.

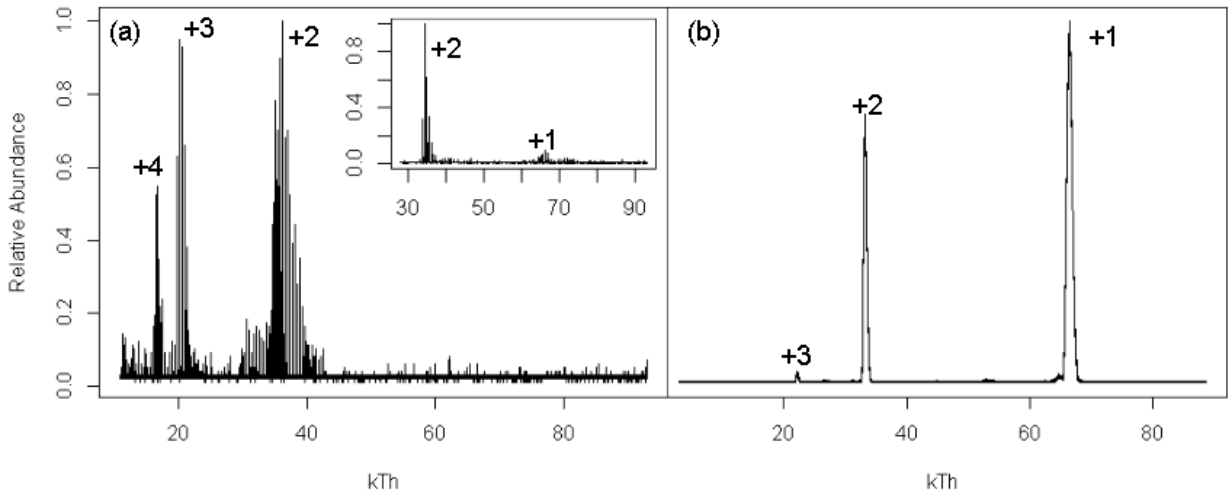

Figure 2.3. Product ions of the BSA and PMD ion/ion reaction measured with (a) resonance ejection and (b) waveform switching. Resonance ejection was performed using a RF ramp of 550 to 5,050 V at 1.008 MHz and a dipolar waveform at 2.2 kHz with a scan length of 100 ms. The spectrum in the insert was measured starting the RF ramp at 2 kV and a scan length of 50 ms. The DIT frequency scan was performed using a ±200 V square wave scanned linearly in *m/z* (nonlinearly in frequency) from 100 to 19 kHz over 50 ms.

Using DIT operation for mass analysis provided ample well depths for high *m/z* ions that were missing in resonance ejection scans at low q_z-values, which allowed us to identify the next limiting factor for high *m/z* performance of an ion/ion reaction in the current QIT. Figure 2.4a shows the post-ion/ion reaction frequency scan for human IgG following a 300 ms ion/ion reaction period. Note that the IgG^{3+}, IgG^{2+}, and IgG^{+} charge states are observed. When the reaction period was extended to 500 ms, the spectrum of Figure 2.4b was obtained. Note that while the IgG^{3+} signal was totally depleted and the IgG^{2+} ion was also largely depleted, the IgG^{+} absolute signal was little changed from the data obtained in Figure 2.4a. This suggests that the mutual storage conditions are marginally effective for storing both the PMD anions and singly charged IgG^{+} ions. The D_r value for the IgG^{+} ion under the mutual storage conditions was 0.4 V, which is the minimum well depth needed to store ions based on our studies with BSA.

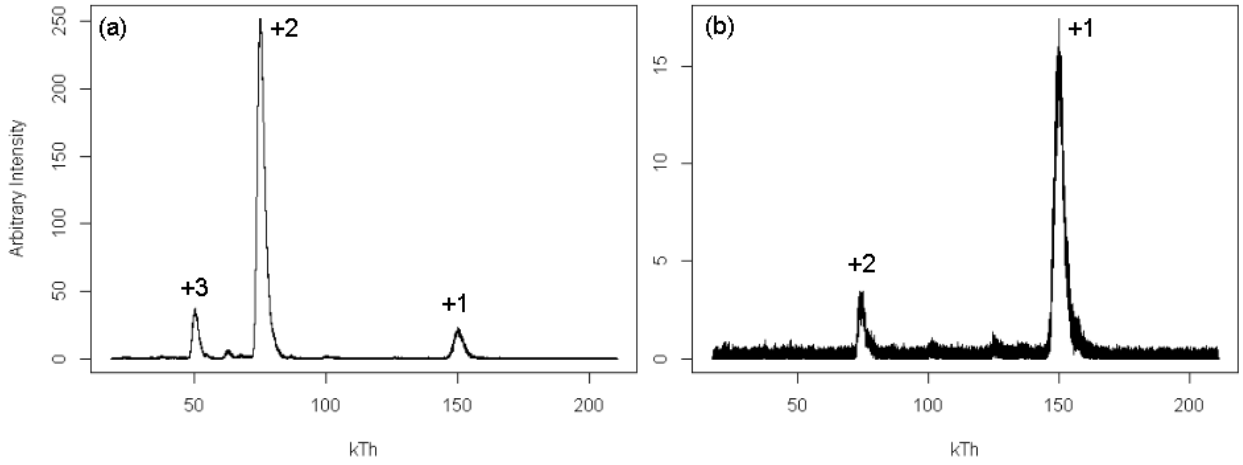

Figure 2.4. Post-ion/ion reaction mass spectrum between PMD anions and human IgG using DIT frequency scanning after (a) 300 ms mutual storage time at an RF voltage of 3,200 V during the reaction period and (b) 500 ms mutual storage time at the same RF voltage. Both spectra were measured using a ±200 V square wave scanned linearly in m/z (nonlinearly in frequency) from 40 to 12 kHz.

2.4 Conclusions

Gas-phase ion/ion reactions can present particularly challenging demands on the *m/z* range of a QIT due to the wide range of ion *m/z* ratios that can be relevant to a particular combination of reactants and products. The charge reduction of initially highly charged bio-ions and bio-ion complexes to ions of relatively low charge states presents such a challenge, especially when the reagent ions used for charge transfer are of low *m/z* ratio. The *m/z* range of a QIT is limited at the low end by the so-called low-mass cut-off, which, in the absence of a DC field, is determined by the *z*-dimension exclusion limit. This is the point at which ions reach $q_z = 0.908$ in a sine wave driven QIT and $q_z = 0.712$ for a square wave driven QIT. The performance of a QIT used as a reaction vessel and analyzer for an ion/ion reaction at high *m/z* values using mass selective instability can be limited by the performance of the detector, the ability to store both reactants and products simultaneously, and the approach used to scan the ions from the ion trap. Resonance ejection at low q_z-values can minimize the RF voltage amplitude needed for ion ejection but leads to shallow well-depths that can result in ion evaporation or an ion cloud size that exceeds the dimensions of the exit aperture of the ion trap. Product ions can be ejected at lower voltages and from deeper well-depths if the RF frequency is reduced. We show here an extension by a factor of 2–3 in upper *m/z* limit via mass selective instability by switching from high frequency sine wave QIT operation to square wave digital ion trap operation with frequency scanning for mass analysis after an ion/ion reaction period. In the present system using DIT operation for mass analysis, the

mutual storage conditions during the ion/ion reaction period becomes the limiting factor in upper *m/z* performance.

2.5 References

[1] R.E. March, J.F. Todd, Quadrupole ion trap mass spectrometry, 2nd edition, Wiley-Interscience, Hoboken, NJ, 2005.

[2] J.N. Louris, J.S. Brodbelt-Lustig, R. Graham Cooks, G.L. Glish, G.J. van Berkel, S.A. McLuckey, Ion isolation and sequential stages of mass spectrometry in a quadrupole ion trap mass spectrometer, Int. J. Mass Spectrom. Ion Process. 96 (1990) 117–137. https://doi.org/10.1016/0168-1176(90)87025-C.

[3] J. V Johnson, R.A. Yost, P.E. Kelley, D.C. Bradford, Tandem-in-Space and Tandem-in-Time Mass Spectrometry: Triple Quadrupoles and Quadrupole Ion Traps, Anal. Chem. 62 (1990) 2162–2172. https://doi.org/10.1021/ac00219a003.

[4] J.N. Louris, R.G. Cooks, J.E. Syka, P.E. Kelley, G.C. Stafford, J.F. Todd, Instrumentation, Applications, and Energy Deposition in Quadrupole Ion-Trap Tandem Mass Spectrometry, Anal. Chem. 59 (1987) 1677–1685. https://doi.org/10.1021/ac00140a021.

[5] J.S. Brodbelt, Photodissociation mass spectrometry: New tools for characterization of biological molecules, Chem. Soc. Rev. 43 (2014) 2757–2783. https://doi.org/10.1039/c3cs60444f.

[6] J.N. Louris, J.S. Brodbelt, R.G. Cooks, Photodissociation in a quadrupole ion trap mass spectrometer using a fiber optic interface, Int. J. Mass Spectrom. Ion Process. 75 (1987) 345–352. https://doi.org/10.1016/0168-1176(87)83045-8.

[7] J.S. Brodbelt, J.J. Wilson, Infrared multiphoton dissociation in quadrupole ion traps, Mass Spectrom. Rev. 28 (2009) 390–424. https://doi.org/10.1002/mas.20216.

[8] F. Vedel, M. Vedel, J.S. Brodbelt, Ion/Molecule Reactions, in: R.E. March, J.F.J. Todd (Eds.), Pract. Asp. Ion Trap Mass Spectrom. Vol. 1, CRC Press, Boca Raton, 1995.

[9] J.B. Fenn, M. Mann, C.K. Meng, S.F. Wong, C.M. Whitehouse, Electrospray ionization for mass spectrometry of large biomolecules, Science (80-.). 246 (1989) 64–71. https://doi.org/10.1126/science.2675315.

[10] R.D. Smith, J.A. Loo, C.G. Edmonds, C.J. Barinaga, H.R. Udseth, New Developments in Biochemical Mass Spectrometry: Electrospray Ionization, Anal. Chem. 62 (1990) 882–899. https://doi.org/10.1021/ac00208a002.

[11] G.J. Van Berkel, G.L. Glish, S.A. McLuckey, Electrospray Ionization Combined with Ion Trap Mass Spectrometry, Anal. Chem. 62 (1990) 1284–1295. https://doi.org/10.1021/ac00212a016.

[12] S.A. McLuckey, J.L. Stephenson, Ion/ion chemistry of high-mass multiply charged ions, Mass Spectrom. Rev. 17 (1998) 369–407. https://doi.org/10.1002/(SICI)1098-2787(1998)17:6<369::AID-MAS1>3.0.CO;2-J.

[13] S.J. Pitteri, S.A. McLuckey, Recent developments in the ion/ion chemistry of high-mass multiply charged ions, Mass Spectrom. Rev. 24 (2005) 931–958. https://doi.org/10.1002/mas.20048.

[14] B.M. Prentice, S.A. Mc Luckey, Gas-phase ion/ion reactions of peptides and proteins: Acid/base, redox, and covalent chemistries, Chem. Commun. 49 (2013) 947–965. https://doi.org/10.1039/c2cc36577d.

[15] J. Stephenson, S.A. McLuckey, Ion/Ion reactions for oligopeptide mixture analysis: Application to mixtures comprised of 0.5-100 kDa components, J. Am. Soc. Mass Spectrom. 9 (1998) 585–596. https://doi.org/10.1016/S1044-0305(98)00025-7.

[16] S.A. McLuckey, G.E. Reid, J. Mitchell Wells, Ion Parking during Ion/Ion Reactions in Electrodynamic Ion Traps, Int. J. Mass Spectrom. Ion Process. 246 (1989) 20. https://doi.org/10.1021/ac0109671.

[17] J.L. Stephenson, S.A. McLuckey, Simplification of Product Ion Spectra Derived from Multiply Charged Parent Ions via Ion/Ion Chemistry, Anal. Chem. 70 (1998) 3533–3544. https://doi.org/10.1021/ac9802832.

[18] M. He, J.F. Emory, S.A. McLuckey, Reagent Anions for Charge Inversion of Polypeptide/Protein Cations in the Gas Phase, J. Am. Soc. Mass Spectrom. 63 (1991) 3173–3182. https://doi.org/10.1021/ac0482312.

[19] L. Konermann, E. Ahadi, A.D. Rodriguez, S. Vahidi, Unraveling the mechanism of electrospray ionization, Anal. Chem. 85 (2013) 2–9. https://doi.org/10.1021/ac302789c.

[20] L. AC, H. AJR, Native Mass Spectrometry: What Is in the Name?, J. Am. Soc. Mass Spectrom. 28 (2017) 5–13. https://doi.org/10.1021/JASMS.8B05378.

[21] G.C. Stafford, P.E. Kelley, J.E.P. Syka, W.E. Reynolds, J.F.J. Todd, Recent improvements in and analytical applications of advanced ion trap technology, Int. J. Mass Spectrom. Ion Process. 60 (1984) 85–98. https://doi.org/10.1016/0168-1176(84)80077-4.

[22] S.A. Mcluckey, G.L. Glish, K.G. Asano, B.C. Grant, Atmospheric Sampling Glow Discharge Ionization Source for the Determination of Trace Organic Compounds in Ambient Air, Anal. Chem. 60 (1988) 2220–2227. https://doi.org/10.1021/ac00171a012.

[23] G.J. Van Berkel, K.G. Asano, P.D. Schnier, Electrochemical processes in a wire-in-a-capillary bulk-loaded, nano-electrospray emitter, J. Am. Soc. Mass Spectrom. 12 (2001) 853–862. https://doi.org/10.1016/S1044-0305(01)00264-1.

[24] D.J. Douglas, A.S. Berdnikov, N. V. Konenkov, The effective potential for ion motion in a radio frequency quadrupole field revisited, Int. J. Mass Spectrom. 377 (2015) 345–354. https://doi.org/10.1016/j.ijms.2014.08.009.

[25] H.G. Dehmelt, Radiofrequency spectroscopy of stored ions I: Storage, in: D.R. Bates (Ed.), Adv. At. Mol. Physics, Vol. 3, Academic, New York, 1967: pp. 53–72.

[26] R.F. Wuerker, H. Shelton, R. V. Langmuir, Electrodynamic containment of charged particles, J. Appl. Phys. 30 (1959) 342–349. https://doi.org/10.1063/1.1735165.

[27] R.E. Kaiser, R. Graham Cooks, G.C. Stafford, J.E.P. Syka, P.H. Hemberger, Operation of a quadrupole ion trap mass spectrometer to achieve high mass/charge ratios, Int. J. Mass Spectrom. Ion Process. 106 (1991) 79–115. https://doi.org/10.1016/0168-1176(91)85013-C.

[28] Z. Nie, F. Cui, M. Chu, C.H. Chen, H.C. Chang, Y. Cai, Calibration of a frequency-scan quadrupole ion trap mass spectrometer for microparticle mass analysis, Int. J. Mass Spectrom. 270 (2008) 8–15. https://doi.org/10.1016/j.ijms.2007.10.012.

[29] D.T. Snyder, C.J. Pulliam, J.S. Wiley, J. Duncan, R.G. Cooks, Experimental Characterization of Secular Frequency Scanning in Ion Trap Mass Spectrometers, 27 (2016) 1243–1255. https://doi.org/10.1007/s13361-016-1377-1.

[30] J.F.J. Todd, A.D. Penman, R.D. Smith, Some alternative scanning methods for the ion trap mass spectrometer, Int. J. Mass Spectrom. Ion Process. 106 (1991) 117–135. https://doi.org/10.1016/0168-1176(91)85014-D.

[31] B.M. Prentice, S.A. Mcluckey, Analysis of High Mass-to-Charge Ions in a Quadrupole Ion Trap Mass Spectrometer via an End-Cap Quadrupolar Direct Current Downscan, Anal. Chem. 84 (2012) 26. https://doi.org/10.1021/ac301741a.

[32] U.P. Schlunegger, M. Stoeckli, R.M. Caprioli, Frequency scan for the analysis of high mass ions generated by matrix-assisted laser desorption/ionization in a Paul trap, Rapid Commun. Mass Spectrom. 13 (1999) 1792–1796. https://doi.org/10.1002/(SICI)1097-0231(19990930)13:18<1792::AID-RCM715>3.0.CO;2-S.

[33] L. Ding, M. Sudakov, F.L. Brancia, R. Giles, S. Kumashiro, A digital ion trap mass spectrometer coupled with atmospheric pressure ion sources, J. Mass Spectrom. 39 (2004) 471–484. https://doi.org/10.1002/jms.637.

[34] L. Ding, M. Sudakov, S. Kumashiro, A simulation study of the digital ion trap mass spectrometer, Int. J. Mass Spectrom. 221 (2002) 117–138. https://doi.org/10.1016/S1387-3806(02)00921-1.

[35] B. Landais, C. Beaugrand, L. Capron-Dukan, M. Sablier, G. Simonneau, C. Rolando, Varying the radio frequency: A new scanning mode for quadrupole analyzers, Rapid Commun. Mass Spectrom. 12 (1998) 302–306. https://doi.org/10.1002/(SICI)1097-0231(19980331)12:6<302::AID-RCM154>3.0.CO;2-U.

[36] N.M. Hoffman, Z.P. Gotlib, B. Opačić, A.P. Huntley, A.M. Moon, K.E.G. Donahoe, G.F. Brabeck, P.T.A. Reilly, Digital Waveform Technology and the Next Generation of Mass Spectrometers, J. Am. Soc. Mass Spectrom. 29 (2018) 331–341. https://doi.org/10.1007/s13361-017-1807-8.

[37] F.L. Brancia, L. Ding, Rectangular waveform driven digital ion trap (DIT) mass spectrometer: Theory and applications, in: R.E. March, J.F.J. Todd (Eds.), Pract. Asp. Trapped Ion Mass Spectrom. Vol. IV Theory Instrum., CRC Press, Boca Raton, 2010.

[38] P.T.A. Reilly, G.F. Brabeck, Mapping the pseudopotential well for all values of the Mathieu parameter q in digital and sinusoidal ion traps, Int. J. Mass Spectrom. 392 (2015) 86–90. https://doi.org/10.1016/j.ijms.2015.09.013.

[39] S.A. McLuckey, J. Wu, J.L. Bundy, J.L. Stephenson, G.B. Hurst, Oligonucleotide mixture analysis via electrospray and ion/ion reactions in a quadrupole ion trap, Anal. Chem. 74 (2002) 976–984. https://doi.org/10.1021/ac011015y.

CHAPTER 3. ION TRAP OPERATIONAL MODES FOR ION/ION REACTIONS YIELDING HIGH MASS-TO-CHARGE PRODUCT IONS

3.1 Introduction

Increased interest in the application of mass spectrometry to large intact biopolymers, mixtures of biopolymers, and biomolecular complexes has motivated the development of technologies that support the formation, analysis, and detection of high mass ions. Electrospray ionization (ESI) is a widely used ion generation technique for biomolecules in part due to its ability to ionize intact species, including non-covalently bound complexes, with minimal fragmentation. A hallmark of ESI is its tendency to form multiply-charged ions from biomolecules over a range of charge states [1]. This characteristic can be both advantageous, as it provides multiple mass measurements in a single spectrum and enables use of analyzers of modest upper *m/z* range, and problematic, as it dilutes signal among multiple charge states and places greater demands on the analyzer to resolve charge states. Ion/ion charge-reduction reactions provide a solution when charge states are unresolved by generating lower charge states that have a higher degree of separation[2,3]. Quadrupole ion traps (QIT) are useful platforms for ion/ion reactions because they enable simultaneous confinement of ions of both polarities over wide ranges of mass-to-charge (*m/z*) ratios. Both QITs and linear electrodynamic ion traps (LITs) have been used as reaction vessels for ion/ion reactions in a variety of applications [2–5]. In some cases, the ion trap is used as the vessel for ion/ion reactions while the products are transferred to a separate analyzer, such as a time-of-flight (TOF) analyzer [6], an Orbitrap™ [7], or an ion cyclotron resonance (ICR) cell [8] for *m/z* measurement. The QIT, however, is capable of analyzing high *m/z* ions in various ways [9–14], which makes it capable of serving as an ion accumulator, ion reactor, and mass analyzer. While the mass analysis figures of merit are modest relative to those of FT-ICR and the Orbitrap™, for example, the FT-ICR resolution decreases as m/z^{-1} and the resolution of the Orbitrap decreases as $m/z^{-1/2}$. At very high *m/z*, therefore, the QIT resolution may be competitive with the other trapping technologies. The resolution of a TOF analyzer is less dependent on *m/z*

but is much larger than a QIT and is highly sensitive to injection conditions. We are therefore interested in exploring the performance of the QIT in ion/ion reaction applications that result in product ions of *m/z* in the tens to hundreds of thousands.

There are two distinct issues that are particularly relevant to the use of a QIT as both reactor and mass analyzer in an ion/ion reaction experiment leading to very high *m/z* ions: 1) the *m/z* range over which both the reactants and products can be stored simultaneously and 2) the *m/z* range over which product ions can be analyzed. The first issue is associated with the mutual storage step of the experiment and reflects the common situation that the analyte ions decrease in charge state, and therefore increase in *m/z*, while the typically singly-charged reagent ions undergo neutralization (i.e., the remaining reagent ions do not change in *m/z*). The second issue, which is focused on the mass analysis step, relates to the range of product ion *m/z* (i.e., both the upper and lower *m/z* limits for a given set of conditions) as well as resolution, efficiency, etc. In this report, we have focused our attention on five QIT operation approaches using resonance ejection for mass analysis [8–13]. Four of the operational modes are based on the conventional use of sine waves for both trapping and resonance ejection while the fifth is based on frequency scanning using digital (i.e., square wave) waveforms [15]. We have recently described a mixed-mode of operation that employed sine wave operation during the ion/ion mutual storage period and square wave frequency scanning for mass analysis [16]. While this mixed-mode operation has some unique advantages, it requires the electronics and control software for both types of operation. Furthermore, it was not straightforward to effect resonance ejection in the frequency scan in the mixed-mode operation. We have therefore restricted our discussion here to either purely sine wave or purely digital operation. We demonstrate here the generation and mass analysis of product ions of several hundred thousand *m/z* for all modes but best overall performance with digital operation.

3.2 Experimental

3.2.1 Materials

Perfluoro-1-octanol (PFO, 400 Da), bovine serum albumin (BSA, 66,430 Da), pyruvate kinase (PK, 232 kDa) from rabbit muscle, and chaperonin 60 (GroEL, ~800 kDa) from *Escherichia coli* were purchased from MilliporeSigma (St. Louis, MO). BSA was dissolved in 100 mM ammonium acetate (10 μM) with no further sample preparation. PK was dissolved in 150 mM

ammonium acetate (5 µM) and filtered using Amicon Ultra-0.5 mL centrifugal filters (MilliporeSigma, St. Louis, MO) with 100 kDa molecular weight cut-off. A 0.5 mL aliquot was placed on the filter and centrifuged at 14,000 G for 10 minutes. The filter was then inverted into a new tube and centrifuged at 1000 G for 2 minutes, and the concentrate was diluted back to 0.5 mL. GroEL was prepared following a published procedure [17]. A 20 µM solution of GroEL was made in buffer A (20 mM tris-HCL, 50 mM potassium acetate, 0.5 mM ethylenediaminetetraacetic acid, 5 mM magnesium chloride) and 2 mM adenosine-5'-triphosphate (ATP) adjusted to pH 7. The solution was shaken slowly for 1 hour followed by addition of methanol (20% of total volume). The solution was again shaken for 1 hour and acetone was added (50% of total volume) to precipitate the protein. The liquid was decanted and the precipitate was dissolved in buffer A with added ATP to a final monomer concentration of 20 µM. The solution was shaken for an hour and filtered using Amicon Ultra-0.5 mL centrifugal filters with 10 kDa molecular weight cut-off four times. The concentrate was diluted back to 20 µM in buffer A with ATP the first three times and in 200 mM ammonium acetate the final time. A final filtration using a 100 kDa molecular weight cut-off filter led to cleaner mass spectra. PFO was dissolved in a 99:1 (v/v/) methanol/ammonium hydroxide solution at a concentration of ~30 mM.

3.2.2 Operational Modes

The five scan modes examined here include: 1) forward sine wave ring electrode amplitude scan with fixed resonance ejection frequency, 2) reverse sine wave amplitude scan with fixed resonance ejection frequency, 3) fixed ring electrode voltage amplitude with scanned resonance ejection frequency (high frequency to low), 4) fixed ring electrode voltage amplitude with scanned resonance ejection frequency (low frequency to high), and 5) digital frequency scan of both ring electrode and resonance ejection voltages (high frequency to low). Modes 1–4 apply the main trapping sine wave voltage to the ring electrode with lower amplitude and frequency axial modulation voltages applied in dipolar fashion to the end cap electrodes (red sine waves in Figure 3.1). For Mode 5, the main trapping square wave voltage is applied to the ring electrode with lower amplitude and frequency square waves applied to the end cap electrodes for resonance ejection (blue square waves in Figure 3.1). (Note that the sine waves and square waves are not applied simultaneously.) The Mathieu stability parameters describe ion stability and motion in an ion trap. During gas-phase reaction and scanning periods, ion traps are often operated on the $a = 0$

line (i.e., no quadrupolar DC field) to maximize the trapping and scanning *m/z* ranges. Detection of high *m/z* ions was facilitated by a guard ring extraction lens (−300 V) and high voltage conversion dynode (−10 kV). The voltage applied to the electron multiplier was −1600 V. Nitrogen (~1 mTorr) was used in the QIT to improve capture and cooling of high *m/z* ions. The experiments were conducted to be as comparable as possible in terms of ion numbers and scan rates. The major differences observed are therefore a result of differences in the ion/ion mutual storage conditions and mass analysis conditions. Each operational mode is described further in turn.

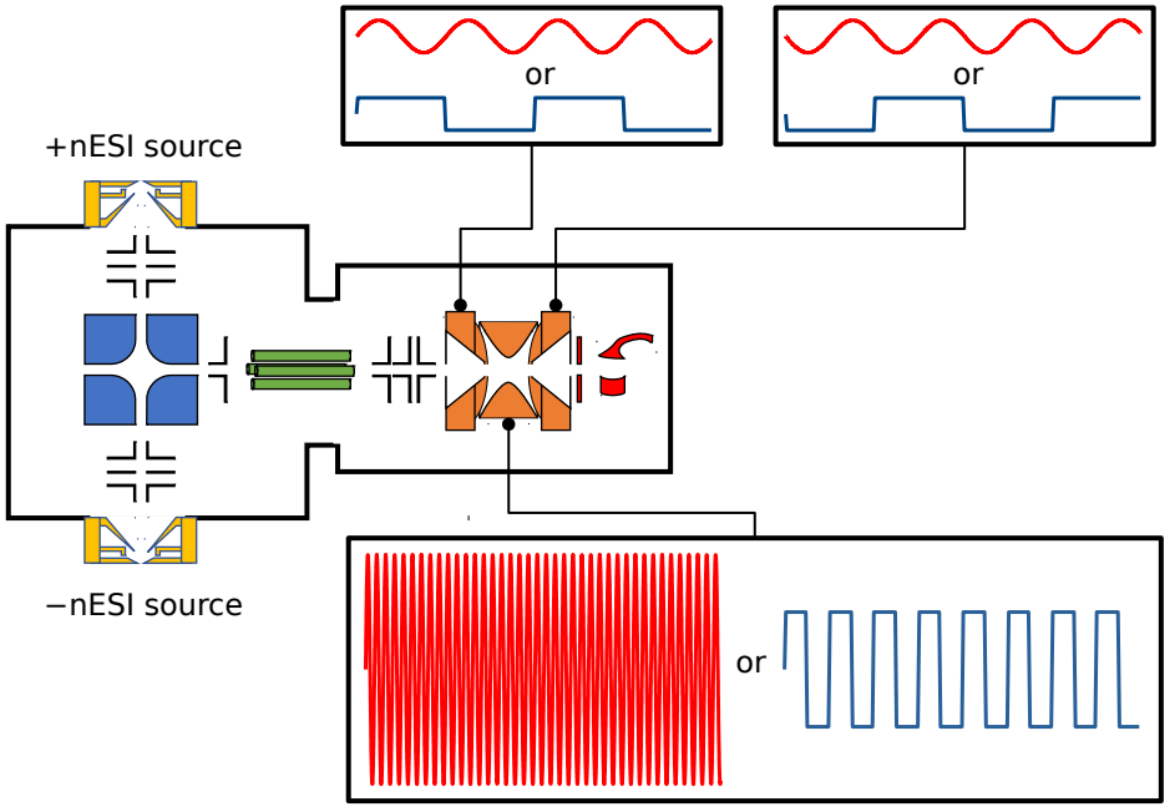

Figure 3.1. Schematic diagram of the home-built 3D ion trap instrument. Red sine waves indicate operation of the ion trap with an amplified high frequency (~1 MHz) sine wave applied to the ring electrode and opposite phases of a low voltage (0.2–10 V) sine wave applied to the end cap electrodes for axial modulation. Blue square waves indicate operation of the ion trap with a lower voltage (±400 V) digital waveform applied to the ring electrode and opposite phases of a low voltage (10 V) square wave applied to the end cap electrodes.

Mode 1

A scan function for a forward sine-wave ring-electrode amplitude scan with fixed resonance ejection frequency at low q-value is shown schematically in Figure 3.2a and represents the 'conventional' approach to mass range extension first described by Kaiser et al. [9] and frequently used for much of the QIT ion/ion reaction research from this laboratory. For this work, the following QIT parameters apply: r_0 = 7.07 mm, Ω = ~1 MHz, and V_{0-p} = 0 to 5 kV. The ion/ion reaction period employed a fixed ring-electrode sine-wave amplitude of 4740 V followed by forward scan of the ring-electrode amplitude with a variable starting voltage with fixed resonance ejection amplitude and frequency. Figure 3.2a also shows a stability diagram with the point along the *q*-axis at which ions are ejected from the ion trap (viz., q_{eject}). The resonance ejection point is sometimes described as a 'hole' in the stability diagram through which ions are ejected into an external detector. In this operational mode, ions with *z*-dimension secular frequencies of motion lower than the resonance ejection frequency (i.e., ions to the left of the hole in the stability diagram) are increased in frequency by increasing amplitude of the ring-electrode voltage such that the ions are translated to the hole (i.e., the q_{eject} point) in the order of low *m/z* to high *m/z*.

Mode 2

A scan function for the reverse sine-wave ring-electrode scan with fixed resonance ejection frequency at low *q*-value is shown schematically in Figure 3.2b, along with a relevant stability diagram. In this case, the ring electrode voltage is increased to the maximum 5 kV at the end of the ion/ion reaction period and scanned down, in contrast with the approach described above. In this case, at the start of the scan the ions of interest have *z*-dimension secular frequencies that are higher than the resonance ejection frequency (i.e., they lie at *q*-values greater than q_{eject}) and the downward scan brings ions into the hole from high *m/z* to low *m/z*.

Mode 3

A scan function for the forward resonance ejection frequency scan is shown in Figure 3.2c, along with a relevant stability diagram. In this case, the ring-electrode voltage remains at 4740 V after the ion/ion reaction period and the resonance ejection frequency is scanned from high to low frequency[12,18]. Scanning the resonance ejection frequency effectively translates the resonance

ejection hole over the ions of interest. To generate a scan that is linear in *m/z* ratio, the frequency was ramped according to its inverse as suggested by the following relationships:

$$q_{eject} \approx \sqrt{2}\beta_{eject} = 2\sqrt{2}\frac{\omega_{eject}}{\Omega} \tag{3.1}$$

$$\frac{m}{z} = \frac{4eV}{q_{eject}r_0^2\Omega^2} \approx \frac{4eV}{2\sqrt{2}r_0^2\Omega\omega_{eject}} \tag{3.2}$$

Mode 4

A scan function for the reverse resonance ejection frequency scan is shown in Figure 3.2d, along with a relevant stability diagram. This mode is identical to Mode 3 with the exception that the resonance ejection frequency is scanned from low frequency to high.

Mode 5

In contrast with the sine-wave operational modes, the digital mode uses a square-wave of fixed amplitude, ±400 V in this case, at variable frequency. The dipolar resonance ejection frequency applied to the end caps is tied to that applied to the ring-electrode so that ions are ejected at a fixed q_{eject}, which for this work was q = 0.5. A scan function for the digital ion trap scan is shown in Figure 3.2e, along with a relevant stability diagram. A constant frequency of 300 kHz is applied during the ion/ion reaction period followed by a downward sweep of the drive and end cap frequencies, thereby resulting in a scan from low *m/z* to high *m/z*. To generate a scan that is linear in *m/z* ratio, the frequency was ramped according to its inverse squared as suggested by the following relationship:

$$\frac{m}{z} = \frac{4eV}{q_{eject}r_0^2\Omega^2} \tag{3.3}$$

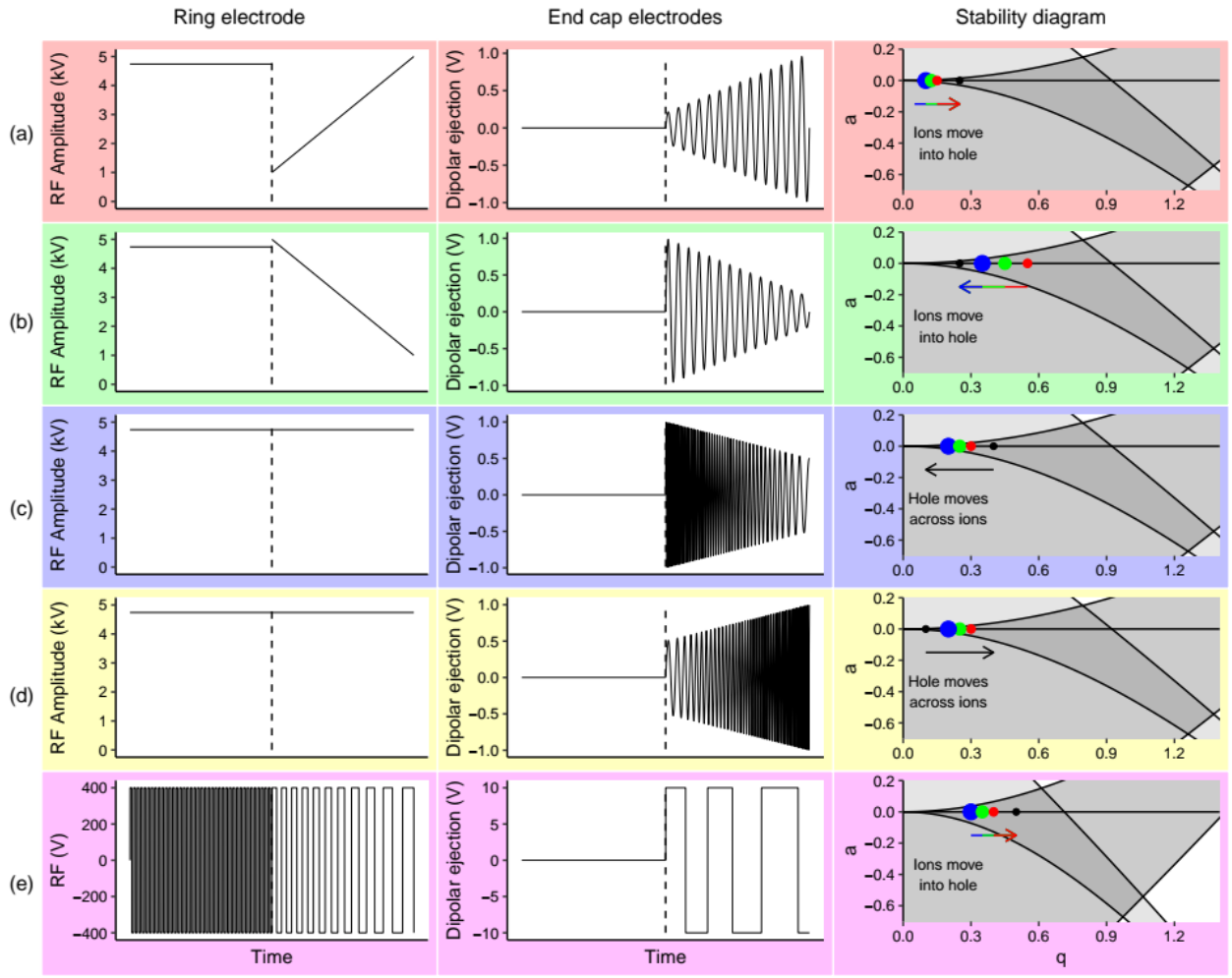

Figure 3.2. Depictions of applied waveforms and associated stability diagrams for the different operational modes. Left of the dotted lines in the waveform plots is the ion/ion reaction period. Right of the dotted line is the scanning period. See text in Section 3.2.2 for descriptions of each mode of operation.

3.3 Results and Discussion

3.3.1 Bovine Serum Albumin

Each of the five operational modes has unique limitations when applied to ion/ion reactions that lead to high *m/z* product ions. However, they all display useful performance up to fairly high *m/z* values using the deprotonated PFO dimer (*m/z* 799) as the reagent anion. Figure 3.3 displays positive ion product ion spectra of BSA charge states generated via proton transfer ion/ion reactions. Figure 3.3a was obtained using Mode 1 by scanning the ring electrode voltage from 1 to 5 kV over 200 ms while increasing the voltage applied to the end caps from 0.2 to 1 V at a constant frequency of 2.8 kHz (q_{eject} = 0.008, scan rate = 290,850 *m/z* s^{-1}). Figure 3.3b was

generated using Mode 2 by scanning the ring electrode voltage from 5 to 0 kV over 250 ms while ramping the end cap voltage from 0.75 to 0 V at the same 2.8 kHz (scan rate = −290852 m/z s^{-1}). The spectra of Figure 3.3c and Figure **3.3**d were collected using Modes 3 and 4, respectively, by scanning the end cap frequency from 16 to 2.5 kHz ($0.045 > q_{eject} > 0.007$) and 2.5 to 16 kHz, respectively, over 250 ms (scan rates = 260,564 m/z s^{-1} and −260,564 m/z s^{-1}, respectively). Figure 3.3e was obtained using Mode 5 by scanning the ring electrode frequency from 90 to 36 kHz over 200 ms while phase locking the end-cap frequency at one-fourth the ring electrode frequency (q_{eject} = 0.5, scan rate = 298,595 m/z s^{-1}). All five modes provided a sufficient trapping potential during the ion/ion reaction period to hold singly-charged BSA until its measurement during the scan period. The most visible difference among the five spectra is the narrower peaks measured by Mode 5. Arguments and models demonstrate that resolution increases with the number of resonance cycles experienced by the ions prior to ejection [19,20]. Resonance ejection frequencies lower than 3 kHz are needed to eject BSA^{1+} using Modes 1 through 4 giving rise to resolutions ($m/\Delta m$) of ~20–40, whereas Mode 5 ejects BSA^{1+} using ~9.3 kHz, primarily due to the much higher q_{eject} value, giving rise to a resolution of ~100. Thus, at the high q_{eject} value achievable by Mode 5, BSA^{1+} experiences more resonance cycles prior to its ejection for any given scan rate. All five modes, however, reproducibly generate and measure singly-charged ions above 50 kDa using the singly-charged PFO dimer reagent anion.

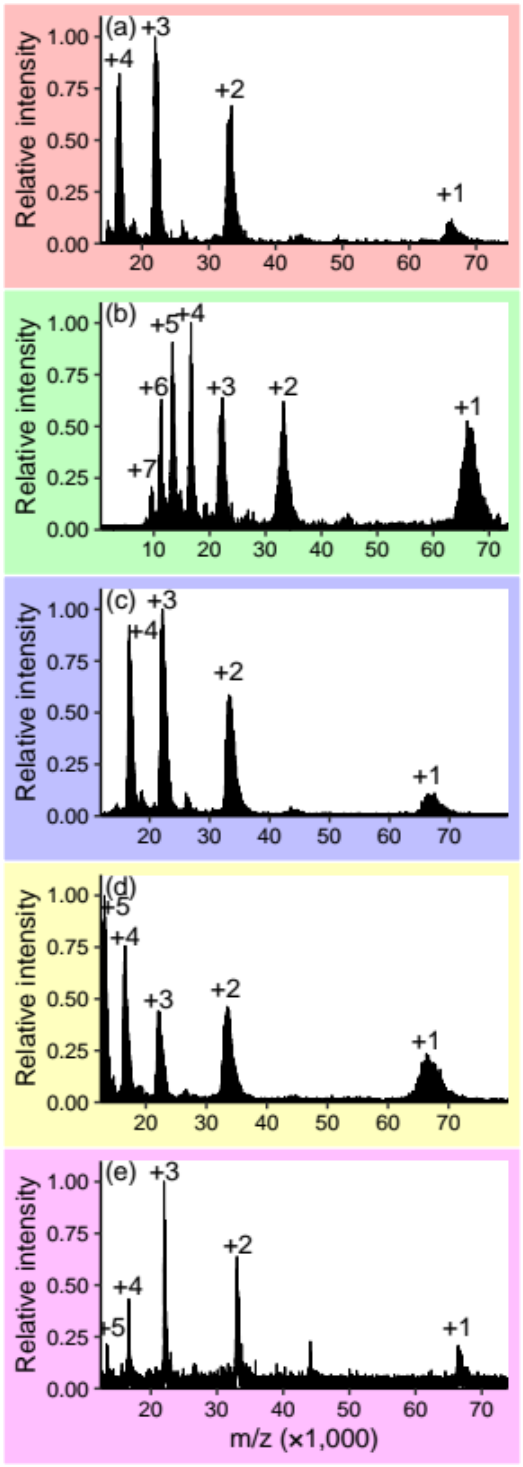

Figure 3.3. Low charge states of BSA measured with different ion trap operational modes: (a) Mode 1, (b) Mode 2, (c) Mode 3, (d) Mode 4, (e) Mode 5. See text in Section 3.3.1 for scan details.

3.3.2 Pyruvate Kinase

Greater performance differences between the various operational modes are noted as product ions increase in *m/z* ratio. Figure 3.4 displays post-ion/ion reaction spectra of PK product ion charge states generated and measured with the five operational modes. Figure 3.4a was collected using Mode 1 by scanning the ring electrode voltage from 1 to 5 kV over 200 ms while ramping the end cap voltage from 0.2 to 1 V at a frequency of 800 Hz (q_{eject} = 0.002, scan rate = 1,017,985 *m/z* s^{-1}). The main limitation of Mode 1 is the requirement to drop the trapping voltage after the reaction period to perform the product ion scan. Starting the scan at 1 kV puts singly-charged PK at a shallow enough well depth (q_z = 0.0005, D_r = 0.031 V) so that ions can escape prior to or early in the scan. The inset to Figure 3.4a shows a ring electrode voltage scan from 2 to 5 kV over 150 ms (end cap voltage was ramped from 0.4 to 1 V at 800 Hz, scan rate = 1,017,980 *m/z* s^{-1}). Singly-charged PK is observed in this spectrum at the expense of a smaller scanning *m/z* range. Figure 3.4b was obtained using Mode 2 by scanning the ring electrode voltage from 5 to 0 kV over 250 ms while ramping the end cap voltage from 0.8 to 0.2 V at the same resonance ejection frequency (scan rate = −1,017,984 *m/z* s^{-1}). Unlike Mode 1, Mode 2 can measure PK^{1+} through PK^{4+} in a single spectrum, but it still exhibits a limited scanning *m/z* range. Because the voltage decreases over the course of the scan, high *m/z* ions are measured first without losing them to evaporation. However, a lower *m/z* scanning limit exists at the point that the trapping voltage scans too low to overcome ion evaporation. Figure 3.4c and Figure **3.4**d were generated using Modes 3 and 4, respectively, by scanning the end-cap frequency from 4.5 to 0.7 kHz (0.013 > q_{eject} > 0.002) and 0.7 to 4.5 kHz, respectively, over 250 ms (scan rates = 931,352 *m/z* s^{-1} and −931,352 *m/z* s^{-1}, respectively). Frequency scanning removes any limitations to the scanning *m/z* range imposed by ramping the trapping voltage. However, limitations associated with a low q_{eject} value still exist. Figure 3.4e was obtained using Mode 5 by scanning the digital trapping frequency from 50 to 19 kHz over 200 ms while phase locking the resonance ejection frequency to one-fourth the frequency of the trapping frequency (q_{eject} = 0.5, scan rate = 1,091,870 *m/z* s^{-1}). Again, being a frequency-based scanning method, the trapping voltage never is dropped or decreased. Similar to BSA, the higher q_{eject} value appears to improve spectral quality because the higher resonance ejection frequency allows the ions to experience more resonance cycles prior to ejection.

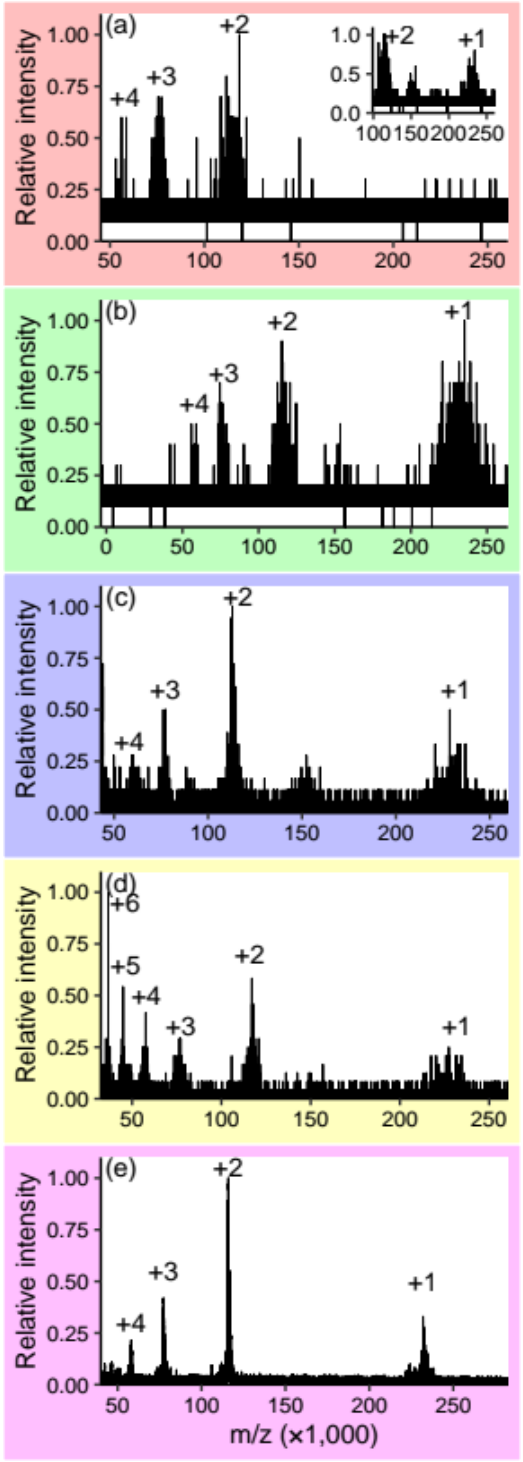

Figure 3.4. Low charge states of PK measured with the different operational modes: (a) Mode 1, (b) Mode 2, (c) Mode 3, (d) Mode 4, (e) Mode 5. See text in Section 3.3.2 for scan details.

3.3.3 GroEL

To further distinguish resonance ejection frequency scanning (Modes 3 and 4) from trapping frequency scanning (Mode 5), low charge states of GroEL were generated via ion/ion reactions and measured. The need for high voltage in voltage scanning modes (i.e., Modes 1 and 2) severely limited their scanning *m/z* ranges and therefore eliminated them as useful modes for measuring ions above *m/z* 200,000. Figure 3.5 contains spectra of GroEL charge states generated and measured via ion/ion reactions with PFO. Figure 3.5a and Figure **3.5**b were generated using Modes 3 and 4, respectively, by scanning the resonance ejection frequency applied to the end caps from 1200 to 150 Hz ($0.003 > q_{eject} > 0.0004$) and 150 to 1200 Hz, respectively, over 250 ms (scan rates = 4,503,568 *m/z* s^{-1} and −4,503,568 *m/z* s^{-1}, respectively). Figure 3.5c was collected using Mode 5 by scanning the digital trapping frequency from 25 to 10.5 kHz over 200 ms while phase locking the resonance ejection frequency on the end caps to one-fourth the trapping frequency ($q_{eject} = 0.5$, scan rate = 3,441,475 *m/z* s^{-1}). GroEL was injected for 1 s (BSA and PK were injected for 200 ms) to improve the probability of maintaining and measuring singly-charged GroEL; however, no significant signal for singly-charged GroEL is visible in any of the spectra. The absence of singly-charged GroEL in Figure 3.5c is readily explained by its shallow well depth potential during the ion/ion reaction ($q_z = 0.0006$, $D_r = 0.027$ V); however, the high voltage sine wave used during the ion/ion reaction in Modes 3 and 4 should provide enough well depth potential ($q_z = 0.0007$, $D_r = 0.20$ V). Decreased detector response at higher *m/z* could explain part of the absence of singly-charged GroEL, but the relative intensity differences in doubly-charged and singly-charged BSA and PK suggest that another factor limits singly-charged GroEL. Likely, the experiment is in a region of the Mathieu stability diagram that is very sensitive to electronic and mechanical imperfections, because of the very low q_z value.

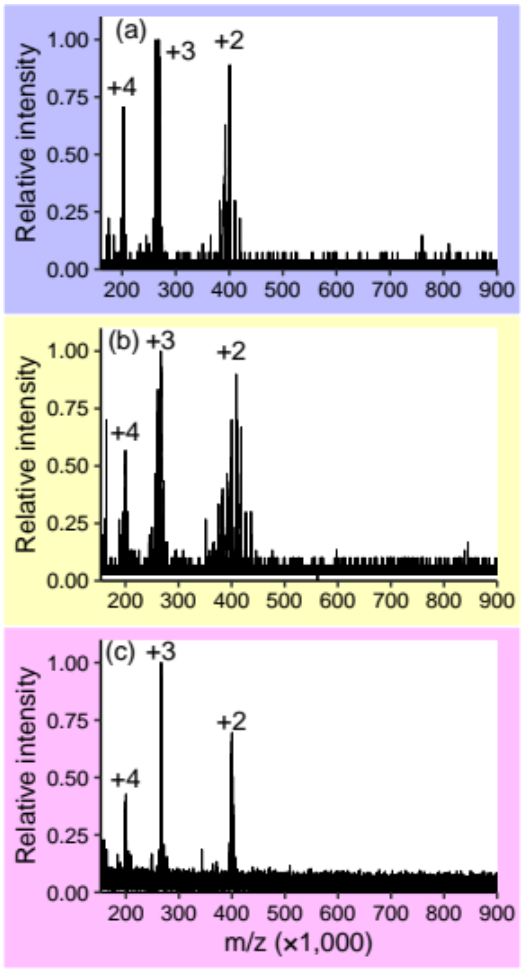

Figure 3.5. Low charge states of GroEL measured with three operational modes. (a) Mode 3, (b) Mode 4, (c) Mode 5. See text in Section 3.3.3 for scan details.

3.3.4 Figures of Merit

Table 3.1–Table **3.5** display calculated q values and well depth potential energies for selected charge states of BSA, PK, and GroEL during the ion/ion reaction and scan periods, as well as at their ejection points for experiments conducted using the five operational modes. Well depth potential energies, defined as the product of ion charge and well depth potential ($z \times D$), describe ion cloud confinement assuming that all ions are thermalized by the gas in the ion trap. Ion cloud size can be approximated by the spring constant of the ions' harmonic motion [21,22]:

$$\kappa = m\omega_r^2 = \frac{2ze}{r_0^2} \times \frac{q_r V}{8} = \frac{2e}{r_0^2}(z \times D_r) \tag{3.4}$$

which is directly proportional to the product of ion charge and well depth potential. Another theoretical approach describes a ratio [23]:

$$\alpha_{E_r} = \frac{zD_{r,eff}}{kT_{eff}} \tag{3.5}$$

that predicts stability of an ion cloud in an ion trap, which is again directly proportional to the product of ion charge and well depth potential. Sine wave well depth can be approximated by $D = 0.125qV$ [24,25] and square wave well depth can be approximated by $D = 0.206qV$ [26] for low q values. Calculated values in Tables S1–S5 suggest that ~0.1 eV of well depth potential energy is enough to trap ions with our instrumental setup, as in the case of measuring singly-charged PK with Mode 5 (Table S5). Additionally, Tables S3 and S4 suggest that ions at a well depth potential energy more than 0.1 eV are lost when at q_z values below ~0.001, as in the case of singly-charged GroEL. Modes 1–4 have better signal-to-noise for the BSA charge states than Mode 5, but similar or worse signal-to-noise for PK and GroEL charge states. In all cases, Modes 1–4 provide deeper trapping potential energies at similar q values during the ion/ion reaction because of the high voltage sine-wave. In the case of BSA, the ejection conditions do not appear to negatively affect the signal, but ejection of PK and GroEL at lower q values do appear to be negatively affected. Mode 5 outperforms all other modes with respect to resolution, most likely due to the q_{eject} value of 0.5, which ejects ions at higher resonance ejection frequencies such that they experience more resonance cycles prior to ejection.

Table 3.1. Comparison of q values and well depth energies ($z \times D_r$) for low charge states of BSA (Figure 3.3a) and PK (Figure 3.4a and inset) during experiments using operational mode 1.

Ion		Reaction		Scan		Ejection		Peak	
Name	*m/z*	q_z	$z \times D_r$ (eV)	$q_{z,min}$	$z \times D_r$ (eV)	q_z	$z \times D_r$ (eV)	S/N	Res.
BSA^{3+}	22,143	0.025	21.82	0.005	0.97	0.008	2.26	311.0	29.9
BSA^{2+}	33,215	0.016	9.70	0.004	0.43	0.008	2.26	207.0	30.7
BSA^{1+}	66,430	0.008	2.42	0.002	0.11	0.008	2.26	35.6	41.5
PK^{3+}	77,334	0.007	6.25	0.002	0.28	0.002	0.64	10.8	11.3
PK^{2+}	116,000	0.005	2.78	0.001	0.12	0.002	0.64	16.6	19.8
PK^{1+}	232,000	0.002	0.69	0.0005 0.001	0.03 0.12	0.002	0.64	N/A 11.9	N/A 26.3

Table 3.2. Comparison of q values and well depth energies ($z \times D_r$) for low charge states of BSA (Figure 3.3b) and PK (Figure 3.4b) during experiments using operational mode 2.

Ion		Reaction		Scan		Ejection		Peak	
Name	*m/z*	q_z	$z \times D_r$ (eV)	$q_{z,min}$	$z \times D_r$ (eV)	q_z	$z \times D_r$ (eV)	S/N	Res.
BSA^{3+}	22,143	0.025	21.82	0.008	2.26	0.008	2.26	129.0	21.5
BSA^{2+}	33,215	0.016	9.70	0.008	2.26	0.008	2.26	125.4	22.1
BSA^{1+}	66,430	0.008	2.42	0.008	2.26	0.008	2.26	105.9	24.9
PK^{3+}	77,334	0.007	6.25	0.002	0.64	0.002	0.64	9.9	17.8
PK^{2+}	116,000	0.005	2.78	0.002	0.64	0.002	0.64	13.5	16.3
PK^{1+}	232,000	0.002	0.69	0.002	0.64	0.002	0.64	15.3	17.2

Table 3.3. Comparison of q values and well depth energies ($z \times D_r$) for low charge states of BSA (Figure 3.3c), PK (Figure 3.4c), and GroEL (Figure 3.5a) during experiments using operational mode 3.

Ion		Reaction		Scan		Ejection		Peak	
Name	*m/z*	q_z	$z \times D_r$ (eV)	$q_{z,min}$	$z \times D_r$ (eV)	q_z	$z \times D_r$ (eV)	S/N	Res.
BSA^{3+}	22,143	0.025	21.82	0.025	21.82	0.025	21.82	770.6	23.7
BSA^{2+}	33,215	0.016	9.70	0.016	9.70	0.016	9.70	450.3	21.8
BSA^{1+}	66,430	0.008	2.42	0.008	2.42	0.008	2.42	82.4	25.9
PK^{3+}	77,334	0.007	6.25	0.007	6.25	0.007	6.25	22.8	40.1
PK^{2+}	116,000	0.005	2.78	0.005	2.78	0.005	2.78	48.4	41.9
PK^{1+}	232,000	0.002	0.69	0.002	0.69	0.002	0.69	22.8	33.9
$GroEL^{3+}$	266,668	0.002	1.81	0.002	1.81	0.002	1.81	53.6	22.3
$GroEL^{2+}$	400,000	0.001	0.81	0.001	0.81	0.001	0.81	47.5	22.8
$GroEL^{1+}$	800,000	0.0007	0.20	0.0007	0.20	0.0007	0.20	N/A	N/A

Table 3.4. Comparison of q values and well depth energies ($z \times D_r$) for low charge states of BSA (Figure 3.3d), PK (Figure 3.4d), and GroEL (Figure 3.5b) during experiments using operational mode 4.

Ion		Reaction		Scan		Ejection		Peak	
Name	*m/z*	q_z	$z \times D_r$ (eV)	$q_{z,min}$	$z \times D_r$ (eV)	q_z	$z \times D_r$ (eV)	S/N	Res.
BSA^{3+}	22,143	0.025	21.82	0.025	21.82	0.025	21.82	233.2	20.2
BSA^{2+}	33,215	0.016	9.70	0.016	9.70	0.016	9.70	244.5	20.9
BSA^{1+}	66,430	0.008	2.42	0.008	2.42	0.008	2.42	122.3	21.5
PK^{3+}	77,334	0.007	6.25	0.007	6.25	0.007	6.25	17.4	26.1
PK^{2+}	116,000	0.005	2.78	0.005	2.78	0.005	2.78	37.3	27.7
PK^{1+}	232,000	0.002	0.69	0.002	0.69	0.002	0.69	14.5	12.5
$GroEL^{3+}$	266,668	0.002	1.81	0.002	1.81	0.002	1.81	58.1	20.6
$GroEL^{2+}$	400,000	0.001	0.81	0.001	0.81	0.001	0.81	51.9	12.7
$GroEL^{1+}$	800,000	0.0007	0.20	0.0007	0.20	0.0007	0.20	N/A	N/A

Table 3.5. Comparison of q values and well depth energies ($z \times D_r$) for low charge states of BSA (Figure 3.3e), PK (Figure 3.4e), and GroEL (Figure 3.5c) during experiments using operational mode 5. D_r values at $q_z = 0.5$ are not accurate thus rough estimates for well depth energies are given.

Ion		Reaction		Scan		Ejection		Peak	
Name	*m/z*	q_z	$z \times D_r$ (eV)	$q_{z,min}$	$z \times D_r$ (eV)	q_z	$z \times D_r$ (eV)	S/N	Res.
BSA^{3+}	22,143	0.023	2.90	0.257	31.75	0.5	~60	96.8	87.0
BSA^{2+}	33,215	0.016	1.29	0.171	14.11	0.5	~40	60.6	91.5
BSA^{1+}	66,430	0.008	0.32	0.086	3.53	0.5	~20	17.9	101.1
PK^{3+}	77,334	0.007	0.83	0.238	29.45	0.5	~60	51.7	108.4
PK^{2+}	116,000	0.004	0.37	0.159	13.09	0.5	~40	126.8	132.6
PK^{1+}	232,000	0.002	0.09	0.079	3.27	0.5	~20	40.1	126.5
$GroEL^{3+}$	266,668	0.002	0.24	0.276	34.16	0.5	~60	67.5	228.0
$GroEL^{2+}$	400,000	0.001	0.11	0.184	15.18	0.5	~40	46.1	198.9
$GroEL^{1+}$	800,000	0.0006	0.03	0.092	3.80	0.5	~20	N/A	N/A

3.4 Conclusions

A home-built QIT instrument provides several approaches to generating and measuring high *m/z* ions via ion/ion reactions, with five technologically available approaches being shown in this publication. Mutual storage of oppositely-charged ions with a high voltage sine wave can be

followed by scanning out ions in forward or reverse directions using an increasing or decreasing voltage ramp of the trapping sine wave applied to the ring electrode (Modes 1 and 2). Product ions can also be scanned in both directions via a decreasing or increasing dipolar resonance ejection frequency ramp applied to the end cap electrodes (Modes 3 and 4). Finally, digital operation performs mutual storage of ions with a lower frequency square wave followed by a decreasing frequency scan applied to the ring electrode with a phase locked dipolar resonance ejection square wave applied to the end cap electrodes (Mode 5). Using a sufficiently high *m/z* reagent ion allows digital operation at lower trapping voltages to compete with traditional sine wave operation at higher voltages. Experiments aimed at generating and measuring low charge states of BSA, PK, and GroEL showed that all modes work well up to *m/z* ~50,000 and that Modes 3–5 work up to *m/z* ~400,000 with Mode 5 generally yielding the best spectra in terms of signal and resolution. Limitations for Modes 1 and 2 are based on the need to decrease trapping voltage during the experiment which creates a trade-off between trapping and scanning *m/z* ranges. Limitations for Modes 3 and 4 result from low q_{eject} values for very high *m/z* ions. Limitations for Mode 5 result from the limited trapping voltage available during the ion/ion reaction to counteract evaporation of very high *m/z* ions at low trapping *q* values.

3.5 References

[1] J.B. Fenn, M. Mann, C.K. Meng, S.F. Wong, C.M. Whitehouse, Electrospray ionization for mass spectrometry of large biomolecules, Science (80-.). 246 (1989) 64–71. https://doi.org/10.1126/science.2675315.

[2] J.L. Stephenson, S.A. McLuckey, Ion/ion proton transfer reactions for protein mixture analysis, Anal. Chem. 68 (1996) 4026–4032. https://doi.org/10.1021/ac9605657.

[3] S.A. McLuckey, J.L. Stephenson, Ion/ion chemistry of high-mass multiply charged ions, Mass Spectrom. Rev. 17 (1998) 369–407. https://doi.org/10.1002/(SICI)1098-2787(1998)17:6<369::AID-MAS1>3.0.CO;2-J.

[4] J.L. Stephenson, S.A. McLuckey, Simplification of Product Ion Spectra Derived from Multiply Charged Parent Ions via Ion/Ion Chemistry, Anal. Chem. 70 (1998) 3533–3544. https://doi.org/10.1021/ac9802832.

[5] M. He, J.F. Emory, S.A. McLuckey, Reagent Anions for Charge Inversion of Polypeptide/Protein Cations in the Gas Phase, J. Am. Soc. Mass Spectrom. 63 (1991) 3173–3182. https://doi.org/10.1021/ac0482312.

[6] Y. Xia, P.A. Chrisman, D.E. Erickson, J. Liu, X. Liang, F.A. Londry, M.J. Yang, S.A. McLuckey, Implementation of ion/ion reactions in a quadrupole/time-of-flight tandem mass spectrometer, Anal. Chem. 78 (2006) 4146–4154. https://doi.org/10.1021/ac0606296.

[7] S.A. Ugrin, A.M. English, J.E.P.P. Syka, D.L. Bai, L.C. Anderson, J. Shabanowitz, D.F. Hunt, Ion-Ion Proton Transfer and Parallel Ion Parking for the Analysis of Mixtures of Intact Proteins on a Modified Orbitrap Mass Analyzer, J. Am. Soc. Mass Spectrom. 30 (2019) 2163–2173. https://doi.org/10.1007/s13361-019-02290-8.

[8] C.R. Weisbrod, N.K. Kaiser, J.E.P. Syka, L. Early, C. Mullen, J.J. Dunyach, A.M. English, L.C. Anderson, G.T. Blakney, J. Shabanowitz, C.L. Hendrickson, A.G. Marshall, D.F. Hunt, Front-End Electron Transfer Dissociation Coupled to a 21 Tesla FT-ICR Mass Spectrometer for Intact Protein Sequence Analysis, J. Am. Soc. Mass Spectrom. 28 (2017) 1787–1795. https://doi.org/10.1007/s13361-017-1702-3.

[9] R.E. Kaiser, J.N. Louris, J.W. Amy, R.G. Cooks, D.F. Hunt, Extending the mass range of the quadrupole ion trap using axial modulation, Rapid Commun. Mass Spectrom. 3 (1989) 225–229. https://doi.org/10.1002/rcm.1290030706.

[10] R.E. Kaiser, R. Graham Cooks, G.C. Stafford, J.E.P. Syka, P.H. Hemberger, Operation of a quadrupole ion trap mass spectrometer to achieve high mass/charge ratios, Int. J. Mass Spectrom. Ion Process. 106 (1991) 79–115. https://doi.org/10.1016/0168-1176(91)85013-C.

[11] L. Ding, M. Sudakov, F.L. Brancia, R. Giles, S. Kumashiro, A digital ion trap mass spectrometer coupled with atmospheric pressure ion sources, J. Mass Spectrom. 39 (2004) 471–484. https://doi.org/10.1002/jms.637.

[12] D.T. Snyder, C.J. Pulliam, J.S. Wiley, J. Duncan, R.G. Cooks, Experimental Characterization of Secular Frequency Scanning in Ion Trap Mass Spectrometers, 27 (2016) 1243–1255. https://doi.org/10.1007/s13361-016-1377-1.

[13] J.D. Williams, K.A. Cox, R.G. Cooks, S.A. McLuckey, K.J. Hart, D.E. Goeringer, Resonance Ejection Ion Trap Mass Spectrometry and Nonlinear Field Contributions: The Effect of Scan Direction on Mass Resolution, Anal. Chem. 66 (1994) 725–729. https://doi.org/10.1021/ac00077a023.

[14] Y. Cai, W. Peng, S. Kuo, Y.T. Lee, H. Chang, Single-Particle Mass Spectrometry of Polystyrene Microspheres and Diamond Nanocrystals The capability of trapping and detection of single microparticles, Quadrupole Storage Mass Spectrom. 88 (1984) 232–238. https://doi.org/10.1021/ac010776y.

[15] N.M. Hoffman, Z.P. Gotlib, B. Opačić, A.P. Huntley, A.M. Moon, K.E.G. Donahoe, G.F. Brabeck, P.T.A. Reilly, Digital Waveform Technology and the Next Generation of Mass Spectrometers, J. Am. Soc. Mass Spectrom. 29 (2018) 331–341. https://doi.org/10.1007/s13361-017-1807-8.

[16] K.W. Lee, G.S. Eakins, M.S. Carlsen, S.A. McLuckey, Increasing the Upper Mass/Charge Limit of a Quadrupole Ion Trap for Ion/Ion Reaction Product Analysis via Waveform Switching, J. Am. Soc. Mass Spectrom. 30 (2019) 1126–1132. https://doi.org/10.1007/s13361-019-02156-z.

[17] M. Zhou, C.M. Jones, V.H. Wysocki, Dissecting the large noncovalent protein complex GroEL with surface-induced dissociation and ion mobility-mass spectrometry, Anal. Chem. 85 (2013) 8262–8267. https://doi.org/10.1021/ac401497c.

[18] D.T. Snyder, C.J. Pulliam, R.G. Cooks, Linear mass scans in quadrupole ion traps using the inverse Mathieu q scan, Rapid Commun. Mass Spectrom. (2016) 2369–2378. https://doi.org/10.1002/rcm.7710.

[19] E. Fischer, Die dreidimensionale Stabilisierung von Ladungsträgern in einem Vierpolfeld, Zeitschrift Für Phys. 156 (1959) 1–26. https://doi.org/10.1007/BF01332512.

[20] J.C. Schwartz, J.E.P. Syka, I. Jardine, High resolution on a quadrupole ion trap mass spectrometer, J. Am. Soc. Mass Spectrom. 2 (1991) 198–204. https://doi.org/10.1016/1044-0305(91)80044-8.

[21] D. Trypogeorgos, C.J. Foot, Cotrapping different species in ion traps using multiple radio frequencies, Phys. Rev. A. 94 (2016) 023609. https://doi.org/10.1103/PhysRevA.94.023609.

[22] C.J. Foot, D. Trypogeorgos, E. Bentine, A. Gardner, M. Keller, Two-frequency operation of a Paul trap to optimise confinement of two species of ions, Int. J. Mass Spectrom. 430 (2018) 117–125. https://doi.org/10.1016/j.ijms.2018.05.007.

[23] P. Delahaye, Analytical model of an ion cloud cooled by collisions in a Paul trap, Eur. Phys. J. A 2019 555. 55 (2019) 1–12. https://doi.org/10.1140/epja/i2019-12740-4.

[24] H.G. Dehmelt, Radiofrequency spectroscopy of stored ions I: Storage, in: D.R. Bates (Ed.), Adv. At. Mol. Physics, Vol. 3, Academic, New York, 1967: pp. 53–72.

[25] R.F. Wuerker, H. Shelton, R. V. Langmuir, Electrodynamic containment of charged particles, J. Appl. Phys. 30 (1959) 342–349. https://doi.org/10.1063/1.1735165.

[26] F.L. Brancia, L. Ding, Rectangular waveform driven digital ion trap (DIT) mass spectrometer: Theory and applications, in: R.E. March, J.F.J. Todd (Eds.), Pract. Asp. Trapped Ion Mass Spectrom. Vol. IV Theory Instrum., CRC Press, Boca Raton, 2010.

CHAPTER 4. DIGITAL ION TRAP MASS ANALYSIS OF HIGH MASS PROTEIN COMPLEXES USING IR ACTIVATION COUPLED WITH ION/ION REACTIONS

Reprinted (adapted) with permission from K.W. Lee, C. P. Harrilal, L. Fu, G.S. Eakins, S.A. McLuckey, Digital Ion Trap Mass Analysis of High Mass Protein Complexes Using IR Activation Coupled with Ion/ion Reactions, Int. J. Mass Spectrom. 458 (2020) 116437. Copyright 2020 Elsevier.

4.1 Introduction

As mass spectrometry expands to include the measurement of large biological complexes, challenges associated with measuring analytes with high mass-to-charge ratios and wide mass distributions have motivated instrumentation developments [1]. Electrospray ionization (ESI) typically generates gas-phase ions of large biomolecules over a range of charge states [2]. The range of charge states permits mass determination using mass analyzers with modest *m/z* ranges [3]. However, ESI also can lead to high levels of solvation and adduction, especially on gas-phase ions generated under 'native' conditions via the charged residue mechanism of ESI [4,5]. High mass gas-phase ions are often generated with high degrees of salt adduction, which leads to poorer peak signal levels and, often, to poor resolution of charge states. Even if individual charge states are visible and confidently assigned, solvation and adduction can significantly affect the accuracy of the mass measurement.

Solution-phase and gas-phase approaches have been employed to mitigate errors associated by solvation and adduction. Chromatography, buffer exchange procedures, and other solution cleaning procedures, for example, can greatly enhance the quality of ESI mass spectra [6,7]. Additionally, instrument adaptions, such as smaller nESI tip diameters, can generate smaller initial droplets leading to gas-phase ions with less initial adduction [8]. The use of relatively heavy background gases and relatively high voltage gradients along the ion path in regions of relatively high pressure has shown to drive off volatile adducts [9,10]. Instruments equipped with ion traps can accumulate and store gas-phase ions for long periods of time and facilitate processes that can further decrease adduction, such as dipolar DC (DDC) activation [11–13], IR activation [14], and ion/ion reactions aimed at removing salts [15,16]. Ion traps also facilitate charge reduction ion/ion reactions that generate lower charge states that are more readily resolved due to the increased *m/z*

spacings between adjacent charge states [17–19]. The combination of charge reduction ion/ion reactions with gas-phase activation thus has the potential to improve both the accuracy of charge state assignments and the accuracy of peak positions on the *m/z* axis leading to more accurate mass measurements of high mass analytes. In this work, we compare DDC and IR activation, two forms of slow heating used in tandem mass spectrometry [20], applied to ions derived from nano-ESI (nESI) of GroEL under 'native' conditions both before and after proton transfer ion/ion reactions. We demonstrate that IR activation provides a greater degree of flexibility than collisional heating (e.g. DDC) in driving off salts and other small adducts, particularly at low charge states.

4.2 Experimental

4.2.1 Materials

Chaperonin 60 from *Escherichia coli* (GroEL, 801 kDa) and perfluoro-1-octanol (PFO, 400 Da) were purchased from MilliporeSigma (St. Lous, MO). GroEL was prepared based on published procedures [21]. 1 mg of GroEL powder was dissolved in 160 µL buffer A (20 mM tris-HCL, 50 mM potassium acetate, 0.5 mM ethylenediaminetetraacetic acid, 5 mM magnesium chloride) with 2 mM adenosine-5'-triphosphate (ATP) adjusted to pH 7. The solution was shaken slowly for 30 minutes. 40 µL ethanol was added to the solution, and it was shaken again for another 30 minutes. 400 µL acetone was added and the solution was allowed to sit for 5 to 15 minutes to allow precipitation of the protein. The mixture was centrifuged at 5,000 g for 5 minutes and liquid was decanted. The pellet was dissolved in 200 µL buffer A with ATP. Buffer exchange into 150 mM ammonium acetate was performed with 100 kDa cut-off Amicon centrifugal filters (MilliporeSigma). The solution was centrifuged at 14,000 g for 10 minutes and washed with 500 µL ammonium acetate at 14,000 g for 10 minutes. The concentrated sample was recovered at 2,000 g for 2 minutes and diluted with ammonium acetate to 500 µL. Positive GroEL ions were generated via nano-ESI (nESI) with 700 to 900 V. PFO was dissolved in 99:1 (v/v) methanol/ammonium hydroxide to a concentration of ~300 µM. Negative singly-deprotonated PFO dimers were generated via nESI with −700 to −800 V.

4.2.2 Instrumentation

An in-house built and modified 3D ion trap mass spectrometer (shown in Figure 4.1) was used for the experiments. The ion trap was operated as a digital ion trap [22,23] using an in-house designed instrument controller and custom electronics [24]. A typical experiment first accumulated positive GroEL ions into the ion trap where they were trapped with a 120 kHz ±400 V square wave applied to the ring electrode. After ramping the frequency to 300 kHz, negative PFO dimer ions (*m/z* = 799) were admitted into the ion trap. Both polarities were stored together for the ion/ion reaction. The frequency was then ramped back to a lower frequency (<120 kHz), depending on the desired *m/z* scan range, to eject unreacted PFO ions and to stabilize high-*m/z* product ions at higher *q* values in preparation for mass analysis. For mass analysis, the frequency was scanned in proportion to the square of its inverse (or linear with *m/z*) to an even lower frequency again depending on the desired scanning *m/z* range. Resonance ejection at *q*=0.5 was effected by applying a dipolar square wave (2.5 to 10 V) to the end cap electrodes that was phase-locked at one-fourth the frequency of the trapping square wave.

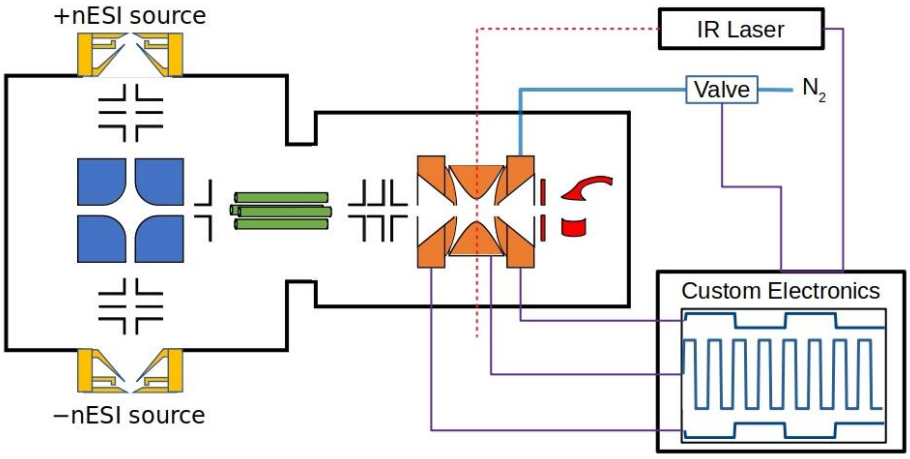

Figure 4.1. 3D digital ion trap mass spectrometer with IR laser and pulsed gas valve.

IR activation was performed using a Synrad (Mukilteo, WA) fan-cooled v40 CO_2 laser (40 W @ 10.6 µm with a beam diameter of 2.5±0.5 mm) that was focused through 1 mm diameter holes in the ring electrode. The supplied laser controller had a TTL input with which it could be gated on and off for a specified amount of time. When on, a built-in command signal (0–5 V @ 5 kHz) triggered firing of the laser. The percentage of laser output was controlled with the supplied

controller that set the duty cycle of the command signal. In this publication, the duty cycle percentage is reported when comparing different laser output percentages. The beam was directed through the holes in the ring electrode using a series of silver coated mirrors (>96% reflectance for 2–20 μm) purchased from Thorlabs (Newton, NJ). The beam was focused to the trap center using a zinc selenide plano-convex lens (>97% average transmission for 7–12 μm) and passed through a BaF_2 wedged window (~70% transmission @ 10.6 μm) into the vacuum chamber. The lens and window were also purchased from Thorlabs.

A TTL controlled pulsed valve was installed in the background gas (N_2) line so that higher pressures (~2 mTorr) could be used to trap and cool incoming ions, and lower pressures (< 1 mTorr) could be used during activation periods to limit collisional cooling [25]. Following the methodology of reference [25], the valve was opened for 5 ms prior to injection of GroEL which filled the trap with nitrogen gas that had built up pressure behind the closed valve during the rest of the previous scan. The initial pulse of gas was enough to efficiently trap both GroEL and PFO for the ion/ion reactions. If the valve was left open during the entire injection period of GroEL (500 ms), longer pump out times were required to reduce the trap pressure for ion activation without providing an increase in ion signal. Pulsing gas prior to mass analysis did not increase signal intensity, possibly because enough residual gas from the initial pulse was present to cool product ions before the mass scan.

4.3 Results and Discussion

4.3.1 DDC vs. IR activation

Figure 4.2 provides a series of product ion spectra involving cations of GroEL in reaction with anions of PFO and illustrates the effect of using either DDC or IR activation coupled with charge reduction ion/ion reactions. (The pre-ion/ion mass spectrum is shown in Figure 4.3a.) The precursor ion population can be activated prior to charge reduction, or the product population can be activated following charge reduction. Prior to activation, the gas valve was closed and ions were trapped for 500 ms so that excess gas could be pumped from the trap to a pressure of approximately 0.8 mTorr to reduce collisional cooling during activation. Without any activation (Figure 4.2a), the relative mass error was +1.9% and the 4+ charge state had an 'apparent resolution' ($m/\Delta m$ FWHM of the envelope of ions at a particular charge state, which includes adducts, isotope

distributions, etc.) of 180. The *m/z* scale for these post ion/ion reaction spectra was calibrated using the charge states of Figure 4.2e assuming a GroEL mass of 801 kDa. Quoted mass errors are therefore relative to any mass errors associated with the scale of Figure 4.2e. For Figure 4.2b, the trapping square wave frequency was decreased to 90 kHz to maximize the DDC voltage that could be applied without losing precursor ions due to the low mass cut-off (LMCO). The maximum DDC voltage without losing ions from the DDC high mass cut-off (HMCO) was 35 V. With 35 V DDC for 100 ms applied before charge reduction, the mass error decreased to +1.1% and the effective resolution of the 4+ charge state increased to 200. Using 40% IR for 100 ms (total fluence of 326 mJ/mm^2) before charge reduction (Figure 4.2c), the mass error further decreased to +0.5% with an effective resolution of 308.

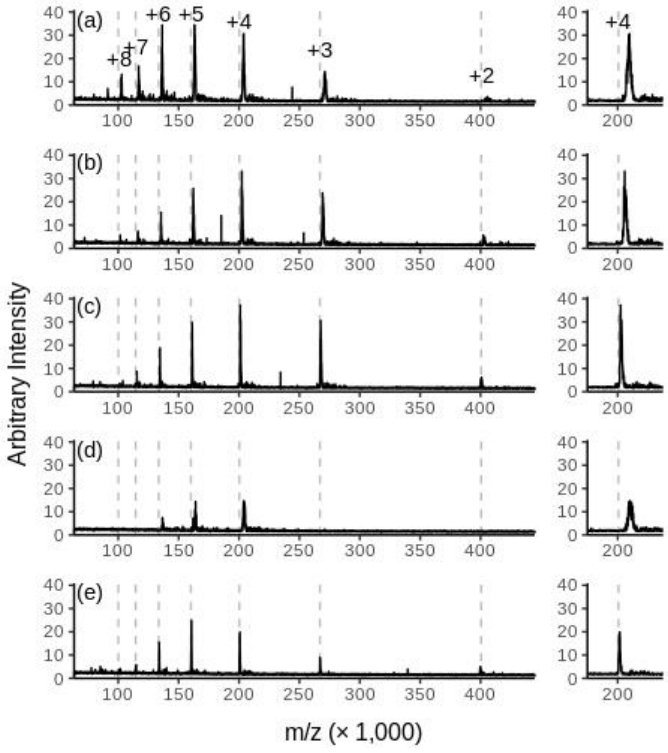

Figure 4.2. Low charge states of GroEL generated via ion/ion reactions with PFO subjected to 100 ms of (a) no activation, (b) 35 V DDC before the ion/ion reaction, (c) 40% IR before the ion/ion reaction, (d) 35 V DDC after the ion/ion reaction, and (e) 30% IR after the ion/ion reaction. A zoomed-in portion highlighting the 4+ charge state is shown to the right of each spectrum. Spectra were collected using a frequency scan from 40 to 15 kHz over 500 ms (scan rate of 762,838 *m/z* s^{-1}) with ions ejected at *q*=0.5 and calibrated using the charge states in (e) with a mass of 801 kDa. Dashed lines indicate the expected *m/z* for the given charge states.

Whereas activation prior to charge reduction (Figure 4.2b-c) simply highlights the ability of IR activation to more effectively overcome collisional cooling and drive off more solvated molecules, given our current DIT electronics, activation following charge reduction (Figure 4.2d-e) illustrates an inherent weakness in DDC activation. To apply 35 V DDC to the product ions following charge reduction without losing all of the low charge states, the trapping square wave frequency was reduced to 23 kHz during DDC activation. The LMCO from the trapping frequency combined with the HMCO from the DDC effectively isolated the 4+ to 6+ charge states (Figure 4.2d); however, the mass error did not decrease (+2.1%) and the effective resolution of the 4+ charge state actually decreased to 122 – likely because of the decrease in signal. A model describing the temperature change due to DDC can explain the apparent inability of DDC to drive off weakly bound adducts in this case. Using a diatomic background gas in a pure quadrupolar trapping field, the dipolar field will increase the ions' temperature by [11]:

$$\Delta T_K = \frac{2m_g}{5k_b m}\langle K_i \rangle \tag{4.1}$$

where m_g is the mass of the background gas, k_b is the Boltzmann constant, m is the mass of the ion, and $\langle K_i \rangle$ is the average ion kinetic energy. Following the derivation in reference [11], but using the well depth potential of a square wave driven 3D ion trap, $D = 0.205qV_{RF}$, the effective potential and effective field across the end cap electrodes are:

$$V(Z) = 0.205q_Z V_{RF}\left(\frac{Z}{Z_0}\right)^2 \tag{4.2}$$

$$E(Z) = -\frac{d}{dZ}V(Z) = -0.410q_Z V_{RF}\frac{Z}{Z_0^2} \tag{4.3}$$

where Z is the axial displacement of the ion of interest, q_Z is the Mathieu q parameter of the ion of interest, V_{RF} is the trapping square wave voltage, and Z_0 is the maximum axial displacement. The equilibrium axial displacement of the ion with an applied dipolar field will satisfy $E_{DDC} = -E(Z_e)$ where Z_e is the equilibrium axial displacement and E_{DDC} is the field due to the applied dipolar DC voltage. The applied dipolar voltage is then $V_{DDC} = 2E_{DDC}Z_0$, and the effective potential at the equilibrium axial displacement can then be written as:

$$V(Z_e) = \frac{V_{DDC}^2}{3.28q_Z V_{RF}} \tag{4.4}$$

where V_{DDC} is the applied DDC voltage. At equilibrium displacement, the average kinetic energy is $\langle K_i \rangle = zeV(Z_e)$ where z is the charge of the ion of interest and e is the fundamental charge. Making this substitution into Equation (4.1) gives:

$$\Delta T_K = \frac{2m_g}{5mk_b}\frac{ze}{3.28q_Z}\frac{V_{DDC}^2}{V_{RF}} = \frac{m_g\Omega^2 r_0^2}{32.8k_b}\left(\frac{V_{DDC}}{V_{RF}}\right)^2 \tag{4.5}$$

where Ω is the trapping frequency, and r_0 is the inner trap radius. Note that for a sine wave driven 3D ion trap, the constant in the denominator of Equation (4.5) would be 20 due to its well depth potential being $D = 0.125qV_{RF}$. In digital trap operation, trapping frequency rather than voltage is variable and determines the temperature increase. Thus, although 35 V DDC was used for both experiments in Figure 4.2b and Figure **4.2**d, the trapping frequency was ~3.9 times greater when activating the precursor ion (Figure 4.2b) than when activating the product ions (Figure 4.2d), leading to a ~15 fold greater temperature increase in the former case relative to the latter heating due to DDC. Using our experimental parameters (m_g = 28 Da, r_0^2 = 0.707 cm), pre-ion/ion DDC activation provided ΔT = 12.6 K (Ω = 90 kHz) and post-ion/ion DDC activation provided ΔT = 0.8 K (Ω = 23 kHz). Unlike DDC activation, IR activation of the products (Figure 4.2e) appears to have a similar effect as activation of the precursor ions. Applying 30% IR activation (total fluence of 244 mJ/mm^2) for 100 ms after charge reduction results in a mass error of 0.1% and 4+ peak resolution of 417. In this case the temperature increase is related to how well the ion cloud interacts with the laser beam. For a well-focused laser beam that is narrower than the ion cloud, a tighter ion cloud leads to greater spatial overlap. Digital operation provides better trapping of high *m/z* ions than traditional sine wave trapping because the trapping frequency can be easily reduced to move high *m/z* ions to high *q* values [24]. The size of an ion cloud can be predicted by its "spring constant" given by [26,27]:

$$Z_{max} = \sqrt{\frac{2E}{\kappa_Z}} = \sqrt{\frac{2E}{m\omega_Z^2}} \tag{4.6}$$

where Z_{max} is the maximum Z-dimension ion displacement, E is the ion kinetic energy, and ω is the ion secular frequency. Assuming that all ions are thermalized to room temperature (~0.04 eV), the average precursor charge state of 64+ trapped with a 90 kHz square wave would have a maximum axial displacement of 24 μm (q_z=0.45, ω_z=20.4 kHz). The 4+ charge state trapped with 40 kHz would have a maximum axial displacement of 170 μm (q_z=0.14, ω_z=2.9 kHz). Both displacements are expected to be less than the beam radius based on the hole (0.5 mm radius) that

the beam passes through. Figure 4.2c and Figure **4.2**e show that IR activation can be equally effective at lower trapping frequencies, unlike DDC activation, because ions can still be focused to the trap center even with very high *m/z* ratios and very low charges. The lower power required to desolvate the low charge states vs. the initial charge states may be related to the fact that the low charge states were activated later in the scan function (following the ion/ion reaction) than the initial charge states, and therefore more background was pumped out prior to activation.

4.3.2 IR fragmentation

Coupling charge reduction ion/ion reactions with an activation approach allows the study of fragmentation patterns over a wide range of different charge states. To observe efficient fragmentation beyond the loss of weakly bound adducts, as noted above, it was necessary to increase the time for excess background gas pump out from 500 ms to 2 s, which reduced the pressure to approximately 0.4 mTorr. Increasing the pump out time beyond 2 s showed no observable difference in complex desolvation and fragmentation. Figure 4.3 illustrates desolvation and fragmentation of native GroEL charge states centered at ~64+ (Figure 4.3a) using 20% IR (Figure 4.3b) and 40% IR (Figure 4.3c), respectively. (Note that Figure 4.2 suggests that with 500 ms pump out prior to IR activation 40% IR power only desolvated the complex without fragmentation of the complex, whereas Figure 4.3 shows that 40% IR power led to extensive dissociation of the complex following 2 s of pump out time.) The *m/z* scale for the spectra in Figure 4.3–Figure **4.8** was calibrated using the charge states of Figure 4.3c assuming an intact GroEL mass of 801 kDa, a monomer mass of 57,214 Da, and a tridecamer mass of 743,786 Da. IR desolvation decreased the relative mass error from +1.3% to +0.1% shown in Figure 4.4a–b. Note that a 1% mass error for the closely spaced native charge states is a much more significant problem than for the widely spaced ion/ion product charge states. The 64+ charge state with no activation measured at *m/z* 12,678 which is closest to the theoretical *m/z* of the 63+ charge state (*m/z* 12,714). After activation, the 64+ charge state measured at *m/z* 12,528 which is closest to the theoretical *m/z* of the 64+ charge state (*m/z* 12,516). This highlights a major benefit for reducing charge via ion/ion reactions to very low charge states. Even with well resolved native charge states, like in Figure 4.3a, a 1% increase in mass due to adducts could very easily lead to incorrect charge state assignments if based purely on matching peak positions to the nearest theoretical *m/z* values [28]. To contrast, the wide separation of the ion/ion product charge states in Figure 4.2a enables

unambiguous charge state assignment even with a mass error due to adducts. Increasing the IR laser power (Figure 4.4c) had little to no further impact on the intact GroEL charge states, but rather induces dissociation of the complex. The resulting highly charged monomer and low charge (n−1)mer from IR multi-photon dissociation (IRMPD) is similar to collision-induced dissociation (CID) of natively sprayed protein complexes [9,10,21,29]. IRMPD of GroEL native charge states ranging from 61+ to 68+ resulted in monomer with charges ranging from 29+ to 40+ and tridecamer with charges ranging from 23+ to 37+, as seen in Figure 4.5.

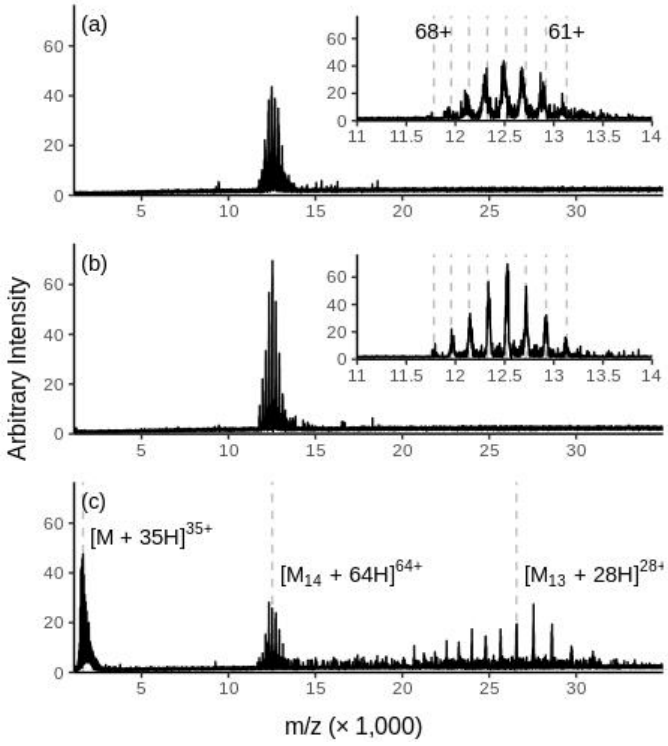

Figure 4.3. Spectra of initial charge states of GroEL with 100 ms of (a) no IR activation, (b) 20% IR activation, and (c) 40% IR activation. Insets of (a) and (b) show zoomed portions of the spectra. Spectra were collected by scanning the trapping frequency from 300 to 45 kHz over 2 s (scan rate of 24,034 m/z s^{-1}) with ions ejected at q=0.5 and calibrated using the spectrum in (c).

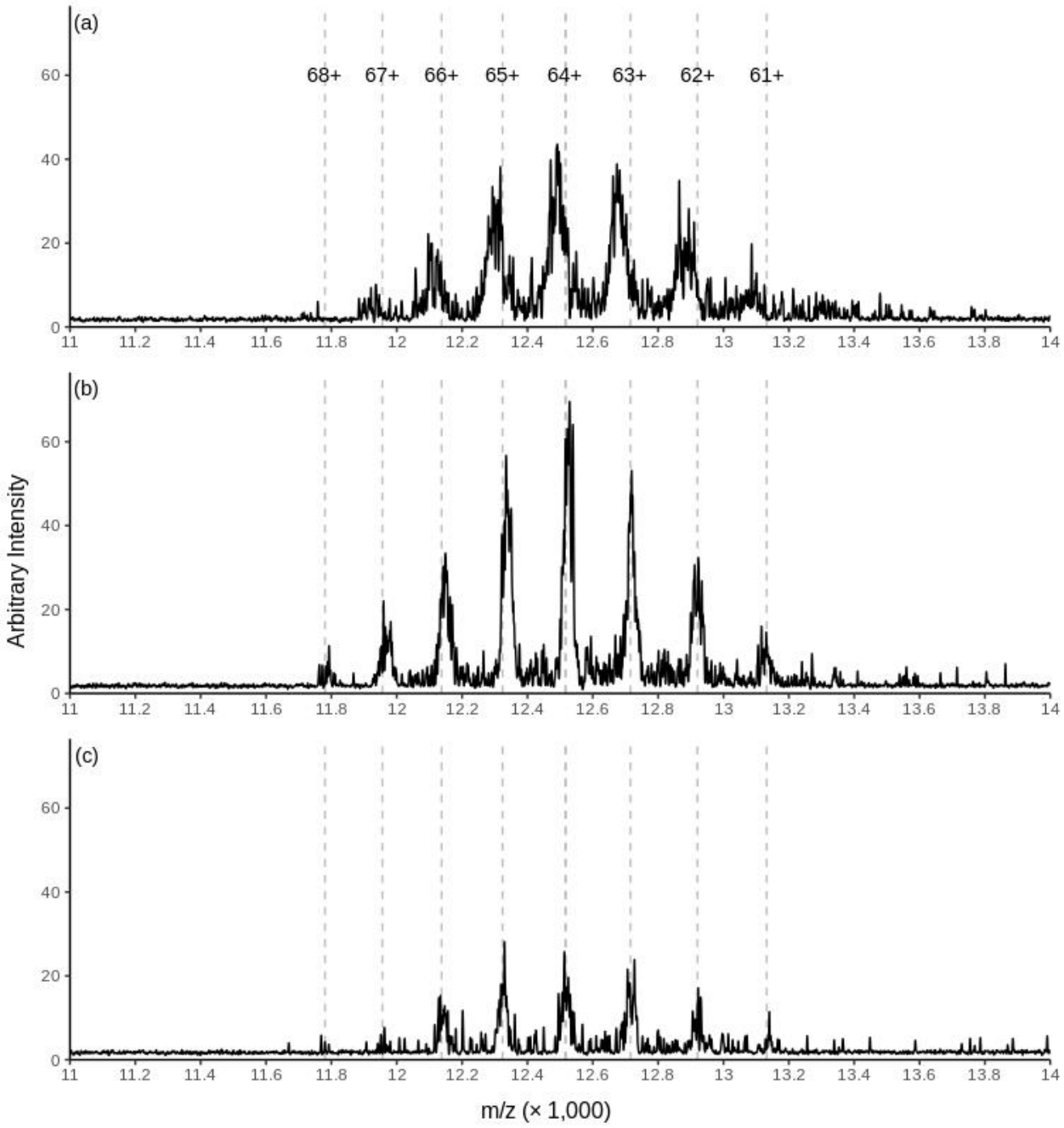

Figure 4.4. Zoomed portions of (a) Figure 4.3a, (b) Figure 4.3b, and (c) Figure 4.3c to illustrate the desolvation effect of IR activation on the native GroEL charge states. Dashed lines indicate the expected *m/z* for the given charge states.

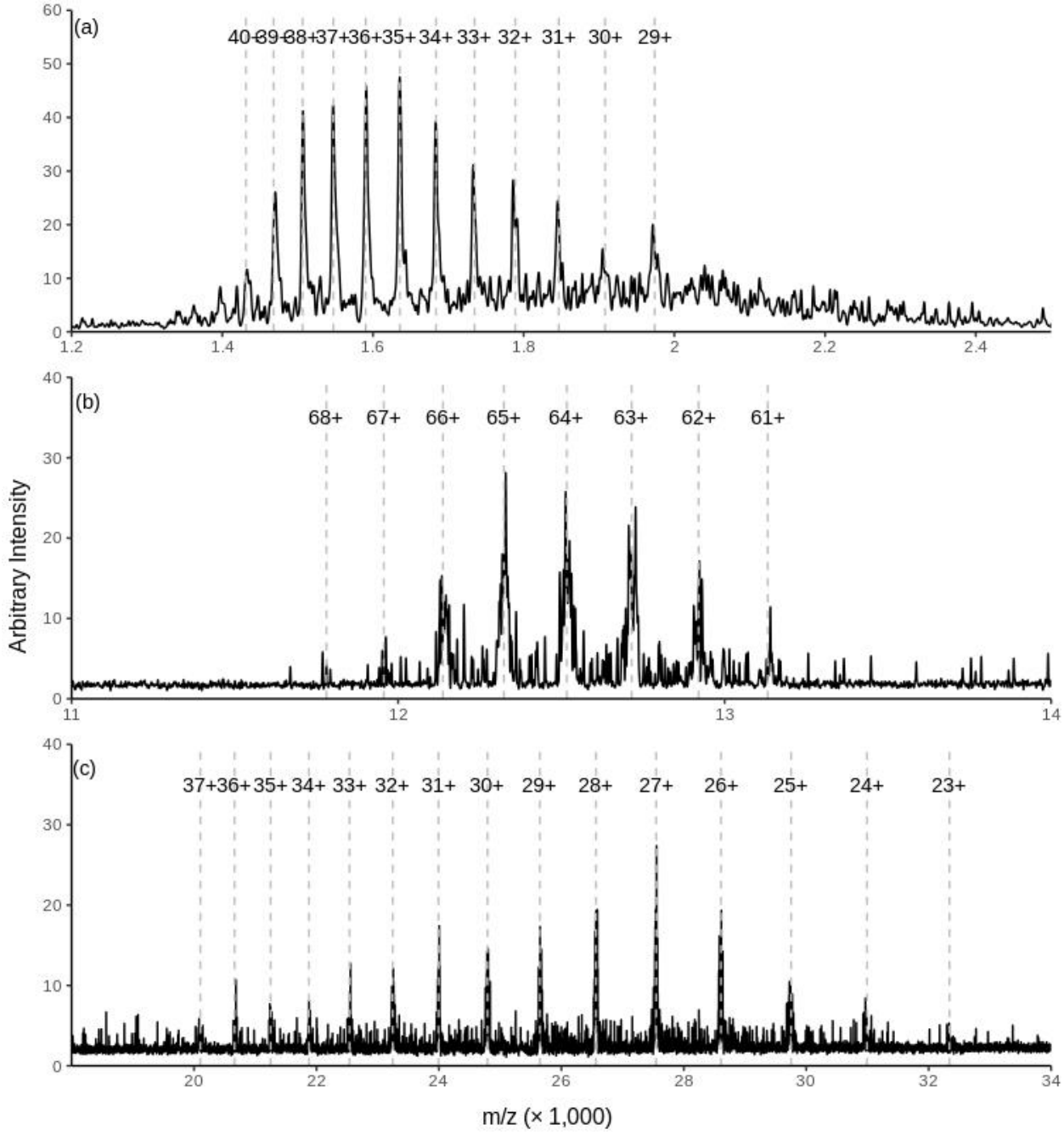

Figure 4.5. Zoomed portions of Figure 4.3c to show (a) monomer, (b) tetradecamer, and (c) tridecamer resulting from IRMPD of native GroEL charge states. Dashed lines indicate the expected *m/z* for the given charge states.

IR activation following a limited amount of charge reduction produced similar results as IR activation of the native charge states, as shown in Figure 4.6. Using 20% IR power led to desolvation with the mass error decreasing from +2.0% (Figure 4.7a) to +0.3% (Figure 4.7b). Again, note that the mass error can lead to incorrect charge state assignment by matching peak

positions to theoretical *m/z* values. Increasing the IR power to 40% further decreased the mass error to +0.1% (Figure 4.7c). Additionally, the complex dissociated into monomer with charges ranging from 18+ to 30+ and tridecamer with charges ranging from 19+ to 23+ as seen in Figure 4.8.

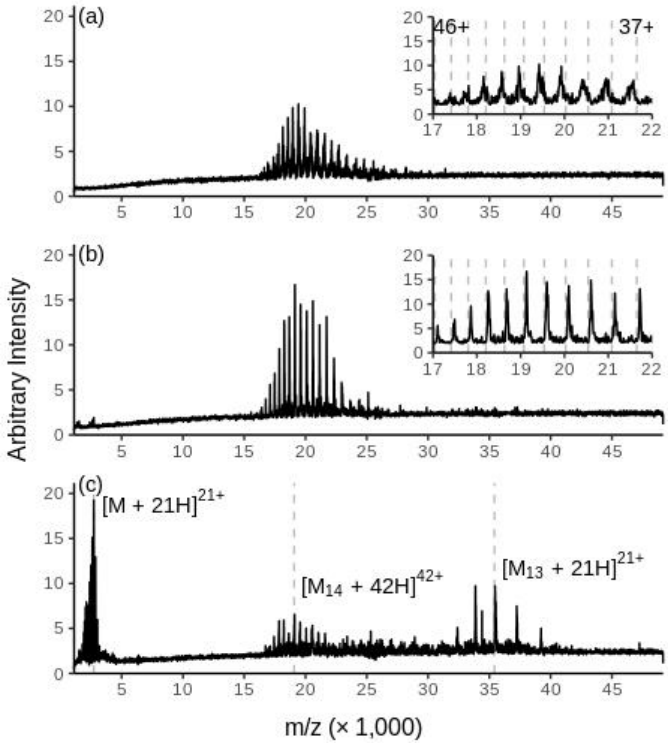

Figure 4.6. Spectra of GroEL charge states centered at ~42+ with 100 ms of (a) no IR activation, (b) 20% IR activation, and (c) 40% IR activation. Insets of (a) and (b) show zoomed portions of the spectra. Spectra were collected by scanning the trapping frequency from 300 to 45 kHz over 2 s (scan rate of 24,034 m/z s^{-1}) with ions ejected at q=0.5 and calibrated using the *spectrum* in Figure 4.3c.

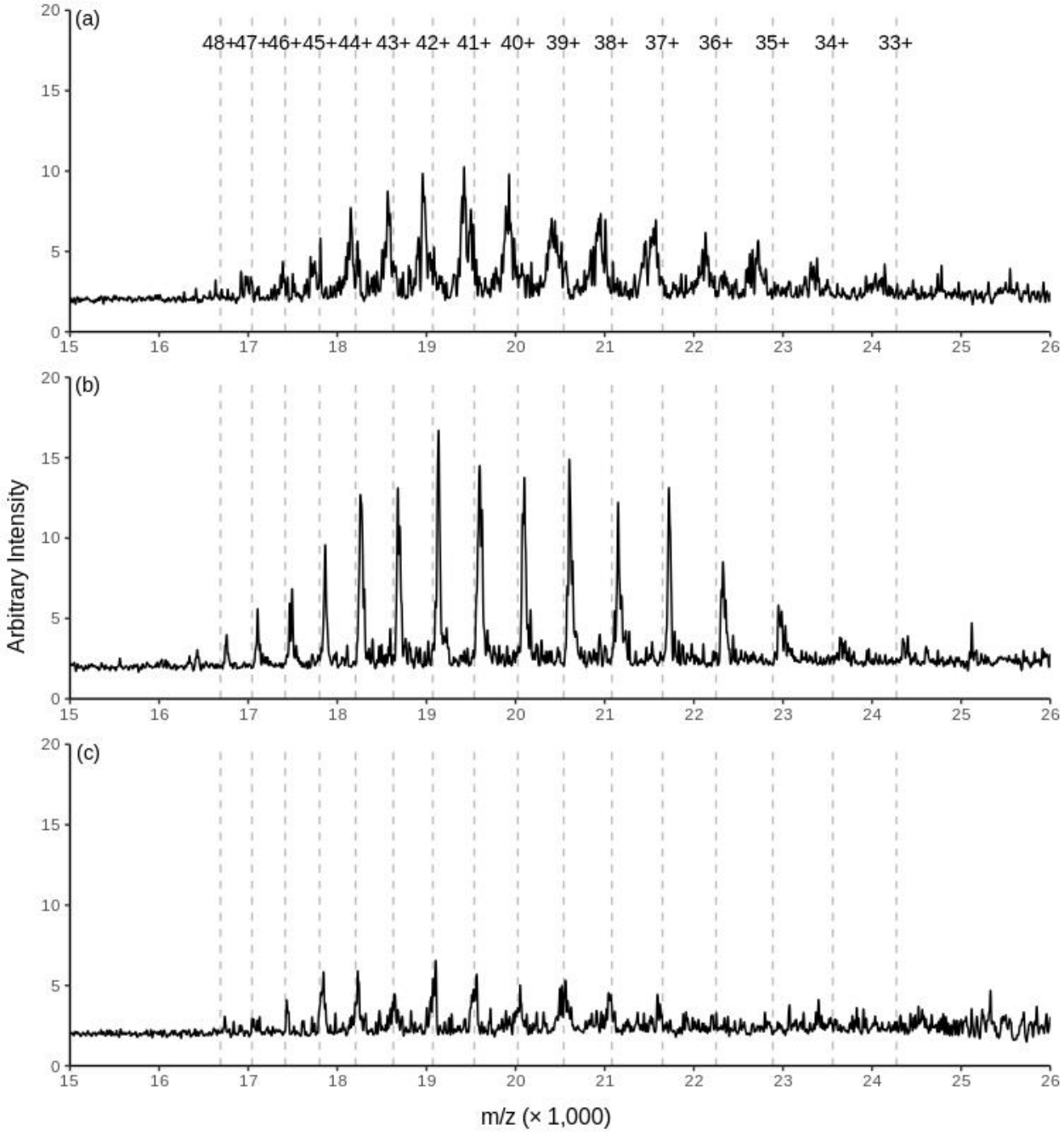

Figure 4.7. Zoomed portions of (a) Figure 4.6a, (b) Figure 4.6b, and (c) Figure 4.6c illustrating the effect of IR desolvation on minimally charged reduced GroEL. Dashed lines indicate the expected *m/z* for the given charge states.

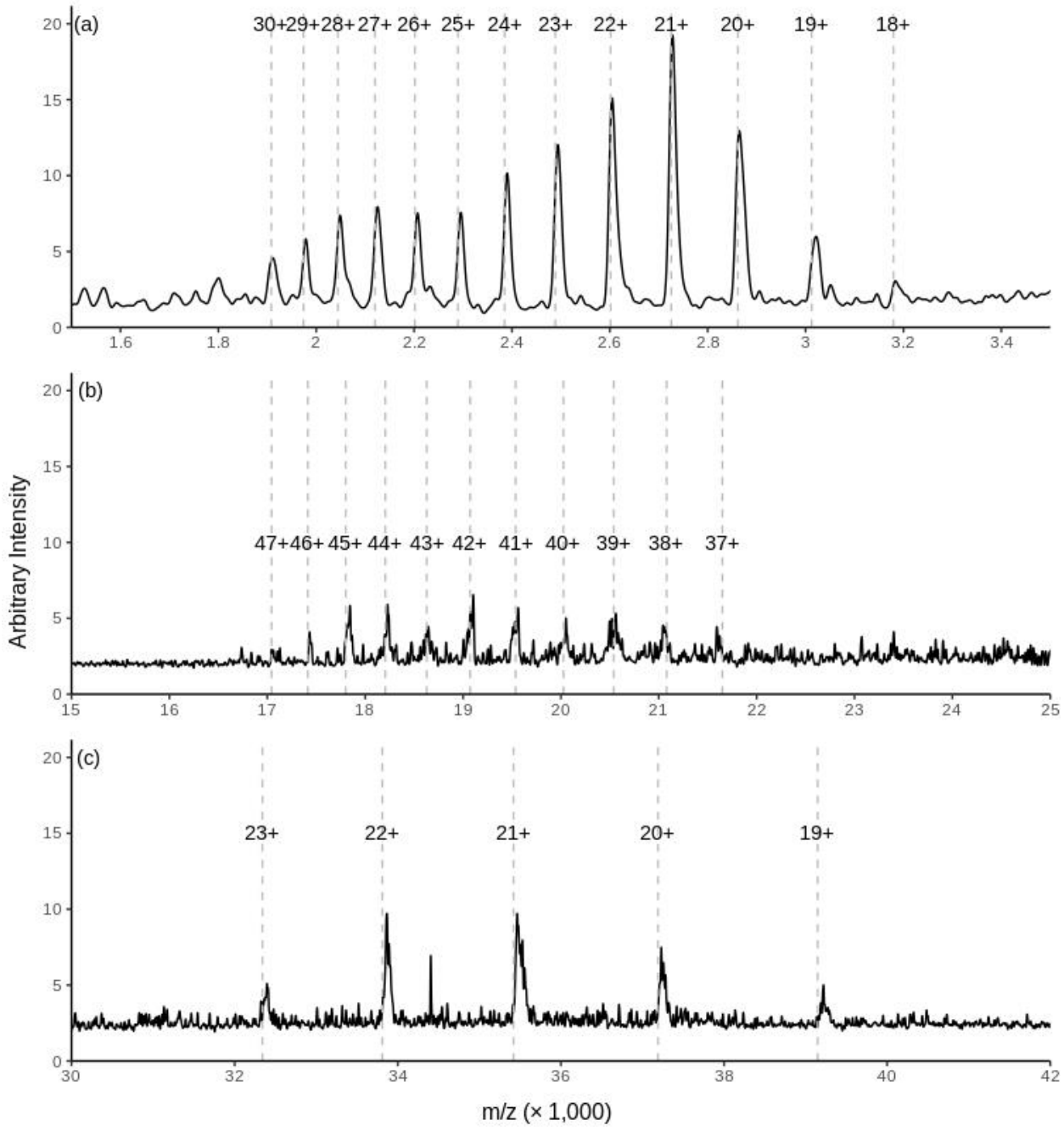

Figure 4.8. Zoomed portions of Figure 4.6c to show (a) monomer, (b) tetradecamer, and (c) tridecamer resulting from IRMPD of minimally charged reduced GroEL. Dashed lines indicate the expected m/z for the given charge states.

When the charge was further reduced to 17+ to 29+, desolvation from 20% IR activation similarly reduced the mass error from +2.0% to +0.3% (seen in Figure 4.9a–b); however, IRMPD using 40% IR activation did not lead to the same pattern of dissociation as with the charge states above 35+. There is no evidence of monomer in the low *m/*z of Figure 4.9c and the charge reduced

tetradecamer charge states are not defined. Previous studies have shown that CID of low charge complexes have stronger inter-unit interactions and collapse to compact structures and/or generate backbone fragments [30–33]. The "blurred" charges states in Figure 4.9c could be explained by many overlapping masses due to a variety of backbone cleavages from the intact complex. New signal also appeared below *m/z* 1500 which could correspond to peptide fragments.

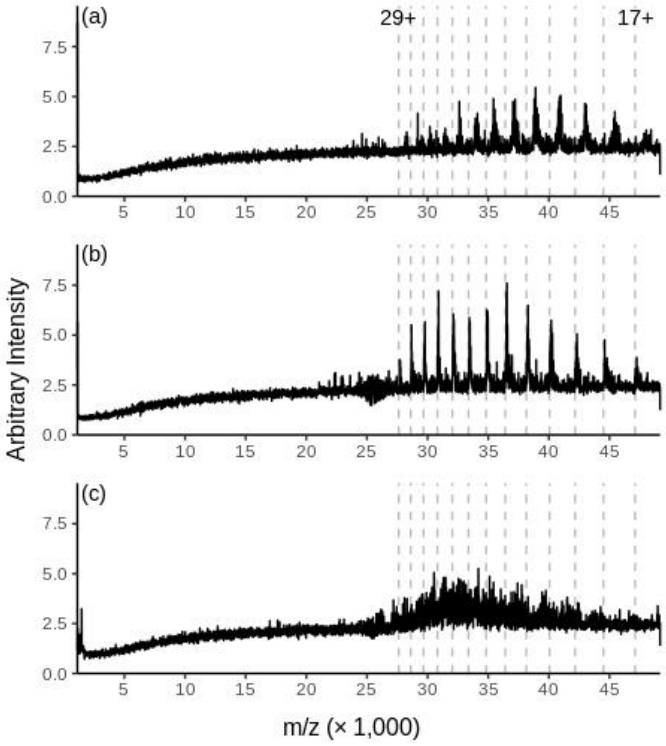

Figure 4.9. Spectra of GroEL charge states centered at ~22+ with 100 ms of (a) no IR activation, (b) 20% IR activation, and (c) 40% IR activation. Spectra were collected by scanning the trapping frequency from 300 to 45 kHz over 2 s (scan rate of 24,034 *m/z* s^{-1}) with ions ejected at q=0.5 and calibrated using the spectrum in Figure 4.3c.

4.4 Conclusions

The combination of gas-phase proton transfer ion/ion reactions with IR activation in a digital ion trap allows for an exceptionally high degree of flexibility in the study of ions of biologically relevant complexes generated via nESI under native conditions. Gas-phase ion/ion reactions can generate any desired charge state lower than the natively sprayed charge states with high efficiency. Digital operation of an ion trap facilitates trapping and focusing of high *m/z* ions allowing, under the conditions used here, for the mass analysis of ions slightly in excess of *m/z* 400,000.

Introducing IR activation with pulsed gas allows for efficient desolvation and fragmentation of the wide range of charge states generated with ion/ion reactions. Digital ion trap operation in conjunction with ion/ion reactions and IR activation has been demonstrated here to allow for the removal of loosely bound adducts from complexes as large as 800 kDa and as low in charge as +2. This combination of technologies therefore shows promise for applications in native MS and native tandem MS, particularly in cases in which extensive peak broadening and charge state overlap arising from extensive salt adduction is observed in the initial native mass spectrum.

4.5 References

[1] A.J.R. Heck, Native mass spectrometry: A bridge between interactomics and structural biology, Nat. Methods. 5 (2008) 927–933. https://doi.org/10.1038/nmeth.1265.

[2] J.B. Fenn, M. Mann, C.K. Meng, S.F. Wong, C.M. Whitehouse, Electrospray ionization for mass spectrometry of large biomolecules, Science (80-.). 246 (1989) 64–71. https://doi.org/10.1126/science.2675315.

[3] M. Mann, C.K. Meng, J.B. Fenn, Interpreting Mass Spectra of Multiply Charged Ions, Anal. Chem. 61 (1989) 1702–1708. https://doi.org/10.1021/ac00190a023.

[4] J. Snijder, R.J. Rose, D. Veesler, J.E. Johnson, A.J.R. Heck, Studying 18 MDa virus assemblies with native mass spectrometry, Angew. Chemie - Int. Ed. 52 (2013) 4020–4023. https://doi.org/10.1002/anie.201210197.

[5] P. Lössl, J. Snijder, A.J.R. Heck, Boundaries of Mass Resolution in Native Mass Spectrometry, J. Am. Soc. Mass Spectrom. 25 (2014) 906–917. https://doi.org/10.1007/s13361-014-0874-3.

[6] A. Laganowsky, E. Reading, J.T.S. Hopper, C. V. Robinson, Mass spectrometry of intact membrane protein complexes, Nat. Protoc. 8 (2013) 639–651. https://doi.org/10.1038/nprot.2013.024.

[7] Z.L. VanAernum, F. Busch, B.J. Jones, M. Jia, Z. Chen, S.E. Boyken, A. Sahasrabuddhe, D. Baker, V.H. Wysocki, Rapid online buffer exchange for screening of proteins, protein complexes and cell lysates by native mass spectrometry, Nat. Protoc. 15 (2020) 1132–1157. https://doi.org/10.1038/s41596-019-0281-0.

[8] A.C. Susa, Z. Xia, E.R. Williams, Small Emitter Tips for Native Mass Spectrometry of Proteins and Protein Complexes from Nonvolatile Buffers That Mimic the Intracellular Environment, Anal. Chem. 89 (2017) 3116–3122. https://doi.org/10.1021/acs.analchem.6b04897.

[9] K. Lorenzen, C. Versluis, E. van Duijn, R.H.H. van den Heuvel, A.J.R. Heck, Optimizing macromolecular tandem mass spectrometry of large non-covalent complexes using heavy collision gases, Int. J. Mass Spectrom. 268 (2007) 198–206. https://doi.org/10.1016/j.ijms.2007.06.012.

[10] R.H.H. Van Den Heuvel, E. Van Duijn, H. Mazon, S.A. Synowsky, K. Lorenzen, C. Versluis, S.J.J. Brouns, D. Langridge, J. Van Der Oost, J. Hoyes, A.J.R. Heck, Improving the performance of a quadrupole time-of-flight instrument for macromolecular mass spectrometry, Anal. Chem. 78 (2006) 7473–7483. https://doi.org/10.1021/ac061039a.

[11] A. V Tolmachev, A.N. Vilkov, B. Bogdanov, L. PĂsa-Tolić, C.D. Masselon, R.D. Smith, Collisional activation of ions in RF ion traps and ion guides: The effective ion temperature treatment, J. Am. Soc. Mass Spectrom. 15 (2004) 1616–1628. https://doi.org/10.1016/j.jasms.2004.07.014.

[12] B.M. Prentice, S.A. Mcluckey, Dipolar DC Collisional Activation in a "Stretched" 3-D Ion Trap: The Effect of Higher Order Fields on rf-Heating, J. Am. Soc. Mass Spectrom. 23 (2012) 736–744. https://doi.org/10.1007/s13361-011-0303-9.

[13] I.K. Webb, Y. Gao, F.A. Londry, S.A. McLuckey, Trapping mode dipolar DC collisional activation in the RF-only ion guide of a linear ion trap/time-of-flight instrument for gaseous bio-ion declustering, J. Mass Spectrom. 48 (2013) 1059–1065. https://doi.org/10.1002/jms.3255.

[14] A. El-Faramawy, Y. Guo, U.H. Verkerk, B.A. Thomson, K.W.M. Siu, Infrared irradiation in the collision cell of a hybrid tandem quadrupole/time-of-flight mass spectrometer for declustering and cleaning of nanoelectrosprayed protein complex ions, Anal. Chem. 82 (2010) 9878–9884. https://doi.org/10.1021/ac102351m.

[15] K.A. Newton, M. He, R. Amunugama, S.A. McLuckey, Selective cation removal from gaseous polypeptide ions: Proton vs. sodium ion abstraction via ion/ion reactions, Phys. Chem. Chem. Phys. 6 (2004) 2710–2717. https://doi.org/10.1039/b315240e.

[16] C.A. Luongo, J. Bu, N.L. Burke, J.D. Gilbert, B.M. Prentice, S. Cummings, C.A. Reed, S.A. Mcluckey, Selective Removal of Alkali Metal Cations from Multiply-Charged Ions via Gas-Phase Ion/Ion Reactions Using Weakly Coordinating Anions, J. Am. Soc. Mass Spectrom. 26 (2015) 404–414. https://doi.org/10.1007/s13361-014-1052-3.

[17] S.A. McLuckey, J.L. Stephenson, Ion/ion chemistry of high-mass multiply charged ions, Mass Spectrom. Rev. 17 (1998) 369–407. https://doi.org/10.1002/(SICI)1098-2787(1998)17:6<369::AID-MAS1>3.0.CO;2-J.

[18] J.L. Stephenson, S.A. McLuckey, Simplification of Product Ion Spectra Derived from Multiply Charged Parent Ions via Ion/Ion Chemistry, Anal. Chem. 70 (1998) 3533–3544. https://doi.org/10.1021/ac9802832.

[19] J.L. Stephenson, S.A. McLuckey, Ion/ion proton transfer reactions for protein mixture analysis, Anal. Chem. 68 (1996) 4026–4032. https://doi.org/10.1021/ac9605657.

[20] S.A. Mcluckey, D.E. Goeringer, Slow Heating Methods in Tandem Mass Spectrometry, 1997.

[21] M. Zhou, C.M. Jones, V.H. Wysocki, Dissecting the large noncovalent protein complex GroEL with surface-induced dissociation and ion mobility-mass spectrometry, Anal. Chem. 85 (2013) 8262–8267. https://doi.org/10.1021/ac401497c.

[22] L. Ding, M. Sudakov, F.L. Brancia, R. Giles, S. Kumashiro, A digital ion trap mass spectrometer coupled with atmospheric pressure ion sources, J. Mass Spectrom. 39 (2004) 471–484. https://doi.org/10.1002/jms.637.

[23] N.M. Hoffman, Z.P. Gotlib, B. Opačić, A.P. Huntley, A.M. Moon, K.E.G. Donahoe, G.F. Brabeck, P.T.A. Reilly, Digital Waveform Technology and the Next Generation of Mass Spectrometers, J. Am. Soc. Mass Spectrom. 29 (2018) 331–341. https://doi.org/10.1007/s13361-017-1807-8.

[24] K.W. Lee, G.S. Eakins, M.S. Carlsen, S.A. McLuckey, Ion trap operational modes for ion/ion reactions yielding high mass-to-charge product ions, Int. J. Mass Spectrom. 451 (2020) 116313. https://doi.org/10.1016/j.ijms.2020.116313.

[25] S.M. Boué, J.L. Stephenson, R.A. Yost, Pulsed helium introduction into a quadrupole ion trap for reduced collisional quenching during infrared multiphoton dissociation of electrosprayed ions, Rapid Commun. Mass Spectrom. 14 (2000) 1391–1397. https://doi.org/10.1002/1097-0231(20000815)14:15<1391::AID-RCM36>3.0.CO;2-O.

[26] D. Trypogeorgos, C.J. Foot, Cotrapping different species in ion traps using multiple radio frequencies, Phys. Rev. A. 94 (2016) 023609. https://doi.org/10.1103/PhysRevA.94.023609.

[27] C.J. Foot, D. Trypogeorgos, E. Bentine, A. Gardner, M. Keller, Two-frequency operation of a Paul trap to optimise confinement of two species of ions, Int. J. Mass Spectrom. 430 (2018) 117–125. https://doi.org/10.1016/j.ijms.2018.05.007.

[28] K.J. Laszlo, M.F. Bush, Analysis of Native-Like Proteins and Protein Complexes Using Cation to Anion Proton Transfer Reactions (CAPTR), J. Am. Soc. Mass Spectrom. 26 (2015) 2152–2161. https://doi.org/10.1007/s13361-015-1245-4.

[29] F. Sobott, C. V. Robinson, Characterising electrosprayed biomolecules using tandem-MS - The noncovalent GroEL chaperonin assembly, Int. J. Mass Spectrom. 236 (2004) 25–32. https://doi.org/10.1016/j.ijms.2004.05.010.

[30] M. Zhou, S. Dagan, V.H. Wysocki, Impact of charge state on gas-phase behaviors of noncovalent protein complexes in collision induced dissociation and surface induced dissociation, Analyst. 138 (2013) 1353–1362. https://doi.org/10.1039/c2an36525a.

[31] Z. Hall, A. Politis, M.F. Bush, L.J. Smith, C. V. Robinson, Charge-state dependent compaction and dissociation of protein complexes: Insights from ion mobility and molecular dynamics, J. Am. Chem. Soc. 134 (2012) 3429–3438. https://doi.org/10.1021/ja2096859.

[32] E.B. Erba, B.T. Ruotolo, D. Barsky, C. V. Robinson, Ion mobility-mass spectrometry reveals the influence of subunit packing and charge on the dissociation of multiprotein complexes, Anal. Chem. 82 (2010) 9702–9710. https://doi.org/10.1021/ac101778e.

[33] K. Pagel, S.J. Hyung, B.T. Ruotolo, C. V. Robinson, Alternate dissociation pathways identified in charge-reduced protein complex ions, Anal. Chem. 82 (2010) 5363–5372. https://doi.org/10.1021/ac101121r.

CHAPTER 5. DEVELOPMENT OF A LINEAR DIGITAL ION TRAP – 3D DIGITAL ION TRAP FOR HIGH MASS ION ATTACHMENT REACTIONS

5.1 Introduction

Native mass spectrometry (MS) is a growing area of research that provides mass and structural information about biomolecular complexes [1,2]. Although current MS technology has facilitated measurements of particles exceeding 10 MDa [3], the inherent heterogeneity leads to complex and congested mass spectra. An additional spectral challenge arises from the commonly-used ionization technique for biomolecules, electrospray ionization (ESI) [4]. ESI generates a range of charge states for a single mass, thus dividing the signal into several distributions along the mass-to-charge (*m/z*) axis and further congesting the spectrum. Single analytes and simple mixtures are readily interpreted through spectral deconvolution [5], and several tools adequately interpret even complex mixtures [6,7]; however, large heterogeneous biomolecular complexes tend to have a high degree of overlap among neighboring charge states which easily leads to misinterpretation.

Ion/ion reactions in an ion trap provide an approach for a "chemical deconvolution" primarily by reducing the charge of ions and thereby increasing the separation between neighboring charge states [8,9]. Proton transfer reactions readily reduce analyte ion charges [10,11]; however, charge reduced biomolecular complexes can exceed the normal mass analysis range of commercial mass spectrometers (>100,000 *m/z*). Digital ion trap (DIT) mass spectrometers readily measure ions in this range because of the flexibility in trapping frequency [12,13]. Because of the power requirements to drive high voltage switches, DIT is limited to lower trapping voltages, which decreases well depth potentials and limits the upper *m/z* range during the ion/ion reaction period [14,15]. A simple solution is to use a higher *m/z* reagent so that a higher low mass cut-off (LMCO) is used during the ion/ion reaction period, which will trap the high *m/z* reaction products in deeper potential wells. Whereas, high mass proton transfer reagents are not readily available, an isolated protein charge state can act as a high mass reagent. Instead of transferring protons, the reagent will complex to the analyte [16,17]. Therefore, whereas a proton transfer reaction generates products that differ in mass and charge by one proton, the protein attachment reaction will generate products that differ in mass and charge by one reagent.

Because a single protein charge state constitutes the reagent ion, high *m/z* isolation is required. The ability to isolate a subset of the analyte mixture can further simplify product spectra by reducing the range of charge states that are analyzed via the ion/ion reaction. DIT also readily provides an approach for high *m/z* isolations through duty cycle manipulation [18,19]. In this work, we demonstrate how an ion/ion reaction between an isolated range of a heterogeneous biomolecular complex and an isolated protein charge state using a dual digital ion trap instrument can facilitate analysis of a biomolecular complex.

5.2 Instrumentation

Figure 5.1 shows a custom-built 3D ion trap mass spectrometer that was modified to include a linear ion trap. Each trap has an independent gas line so that trap pressures can be different. Positive and negative ions are introduced to the mass spectrometer through separate nano-ESI (nESI) interfaces. The digital linear ion trap can accumulate one polarity and perform isolation via duty cycle and frequency manipulation using custom-built electronics. Both polarities can be accumulated and mutually stored in the 3D digital ion trap to perform an ion/ion reaction. Mass analysis is effected by scanning the trapping frequency from high frequency to low according the square of the period such that a linear *m/z* scan is measured.

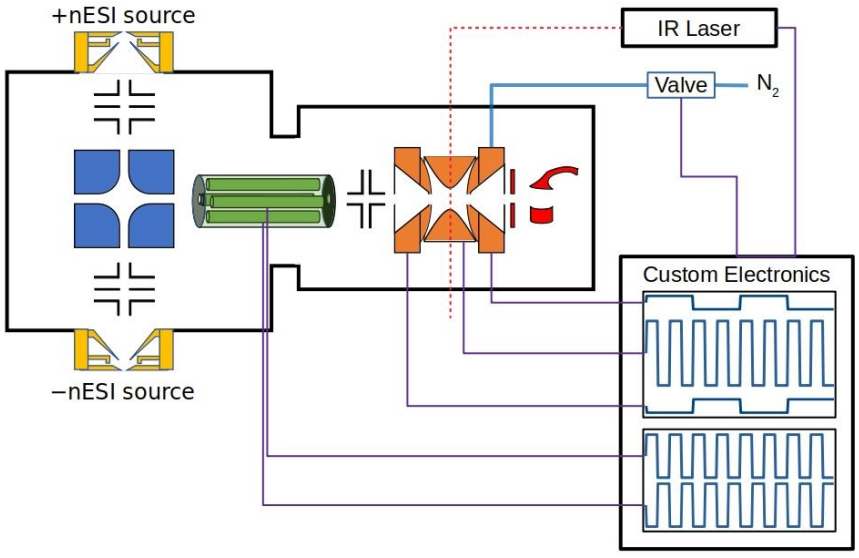

Figure 5.1. Modified 3D ion trap mass spectrometer with added linear ion trap (green rods). Custom electronics are used to operate both the linear and 3D ion traps as digital ion traps.

5.3 Isolation in a Linear Digital Ion Trap

In a more typical sine wave driven linear quadrupole, isolation is accomplished by moving the *m/z* of interest into the apex of the stability region using a quadrupolar DC offset applied to opposing rod pairs. When operated as a mass filter, ions continuously enter the quadrupole, and only those whose *m/z* fall within the stability range defined by the apex pass through the quadrupole [20]. When operated as an ion trap, the quadrupole first traps a wide range of ions using no quadrupolar DC and then applies quadrupolar DC to move the *m/z* range of interest to the stability apex to eject unwanted ions that were previously trapped [21]. Mass filter operation is faster because it eliminates the need to accumulate prior to isolation, but the applied quadrupolar DC creates fringing fields that lower ion acceptance and transmission [22]. This fringing field effect is mitigated by inserting a short quadrupole before the mass filter that has only the sine wave (but no quadrupolar DC) applied to it. Ion trap isolation overcomes the fringing fields by first accumulating ions with no quadrupolar DC. In either case, the maximum *m/z* that can be isolated depends on the maximum achievable sine wave amplitude.

Digital operation of a linear quadrupole overcomes the limited isolation *m/z* range by varying frequency rather than amplitude and provides a unique method for isolation by replacing application of quadrupolar DC with duty cycle modulation [18,19]. Duty cycle modulation effectively applies a quadrupolar DC offset, and therefore creates similar fringing fields as in sine wave operation. These fringing fields are not as readily mitigated because the square wave and its duty cycle are not individually defined voltages as with the sine wave and quadrupolar DC. A linear digital ion trap however can perform a similar isolation as the sine wave linear ion trap by first accumulating ions with a 50% duty cycle square wave and then isolating the region of interest by changing the duty cycle. Figure 5.2 shows the stability region for a digital linear quadrupole with arrows depicting mass filter operation (orange) and ion trap operation (purple).

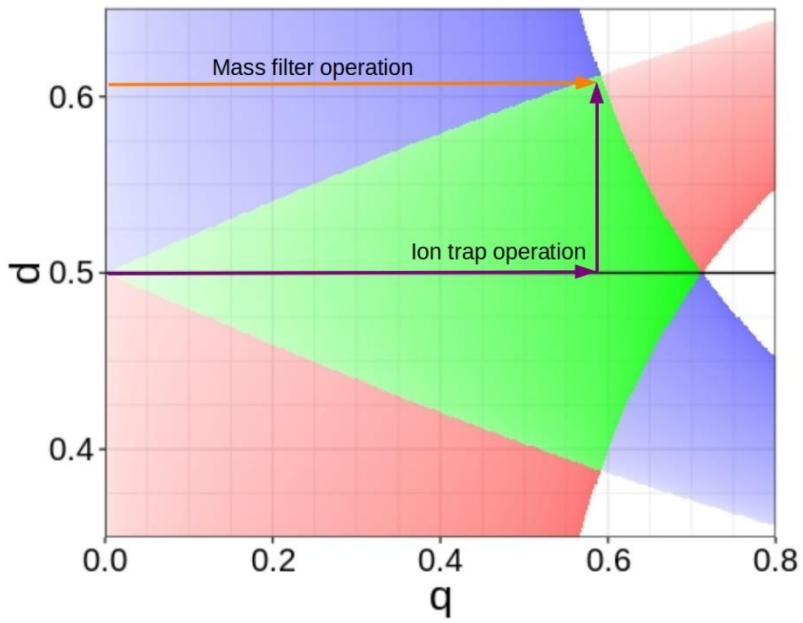

Figure 5.2. Stability diagram of a digital linear quadrupole with duty cycle as the *y*-axis. The orange arrow depicts mass filter operation where ions feel fringing field effects. The purple arrows depict ion trap operation, which eliminates fringing field effects by first trapping a wide *m/z* range at 50% duty cycle and then isolating the *m/z* of interest.

5.4 Simulations of Ion/ion Reaction Spectra

The deconvolution effect of ion/ion reactions can be shown by calculations. Figure 5.3a shows a calculated spectrum for two major components of the 30S *e. Coli* ribosome [23]. The purple and green traces show the underlying charge states for each component, and the black trace shows the sum of both components and illustrates what the mass spectrometer would measure. With only the black trace, a single mass could be determined with relatively low confidence. Isolation of the most abundant charge states (shown in Figure 5.3b) followed by a proton transfer ion/ion reaction (shown in Figure 5.3c) can separate the two masses in the *m/z* range of 40,000 to 45,000, but they overlap again beyond *m/z* 50,000. Eventually charge states of the two components will separate enough in *m/z* space to determine their masses with high confidence, but the total analyte signal will be spread over many peaks with low signal-to-noise. In contrast, performing an ion/ion reaction with the 6− charge state of ubiquitin (shown in Figure 5.3d) generates well separated distributions of charge states. After attachment of four ubiquitin ions, the most abundant charge states of each component are baseline resolved. Further attachments serve to provide more *m/z* measurements that increase the accuracy of the mass measurement by averaging.

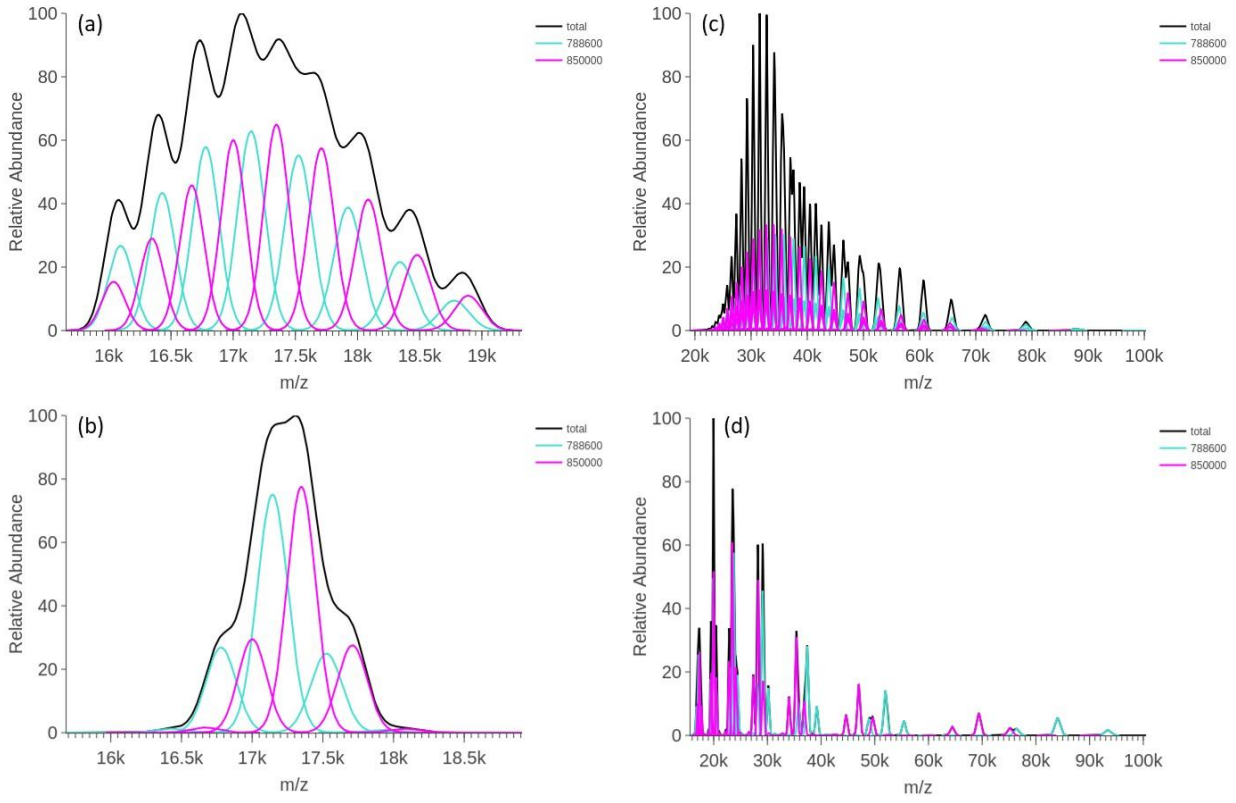

Figure 5.3. Simulation of (a) initial population of two overlapping masses with several charge states each, (b) isolation of most abundant charge states, (c) proton transfer reaction to decrease charge of isolated charge states, (d) reaction with simulated ubiquitin 6− charge state.

5.5 Conclusions

An added digital linear ion trap prior to the 3D digital ion trap provides added experimental flexibility. The linear trap can accumulate and isolate *m/z* ranges of interest using duty cycle modulation of the trapping square wave. Whereas sine wave operation has an upper *m/z* limit for isolation based on the maximum amplitude, digital operation has no theoretical limit. Based on calculations, isolated distributions of heterogeneous biomolecules reacted with an isolated protein charge state generate ion/ion reaction products that are well separated in *m/z* and facilitate identification and mass measurements of multiple components. The combined high mass isolation and mass analysis benefits of digital ion traps and chemical deconvolution strategy of ion/ion reactions can improve analysis of large heterogeneous biomolecular complexes.

5.6 References

[1] A.J.R. Heck, Native mass spectrometry: A bridge between interactomics and structural biology, Nat. Methods. 5 (2008) 927–933. https://doi.org/10.1038/nmeth.1265.

[2] L. AC, H. AJR, Native Mass Spectrometry: What Is in the Name?, J. Am. Soc. Mass Spectrom. 28 (2017) 5–13. https://doi.org/10.1021/JASMS.8B05378.

[3] J. Snijder, R.J. Rose, D. Veesler, J.E. Johnson, A.J.R. Heck, Studying 18 MDa virus assemblies with native mass spectrometry, Angew. Chemie - Int. Ed. 52 (2013) 4020–4023. https://doi.org/10.1002/anie.201210197.

[4] J.B. Fenn, M. Mann, C.K. Meng, S.F. Wong, C.M. Whitehouse, Electrospray ionization for mass spectrometry of large biomolecules, Science (80-.). 246 (1989) 64–71. https://doi.org/10.1126/science.2675315.

[5] M. Mann, C.K. Meng, J.B. Fenn, Interpreting Mass Spectra of Multiply Charged Ions, Anal. Chem. 61 (1989) 1702–1708. https://doi.org/10.1021/ac00190a023.

[6] S.P. Cleary, A.M. Thompson, J.S. Prell, Fourier Analysis Method for Analyzing Highly Congested Mass Spectra of Ion Populations with Repeated Subunits, Anal. Chem. 88 (2016) 6205–6213. https://doi.org/10.1021/acs.analchem.6b01088.

[7] M.T. Marty, A.J. Baldwin, E.G. Marklund, G.K.A. Hochberg, J.L.P. Benesch, C. V. Robinson, Bayesian deconvolution of mass and ion mobility spectra: From binary interactions to polydisperse ensembles, Anal. Chem. 87 (2015) 4370–4376. https://doi.org/10.1021/acs.analchem.5b00140.

[8] S.A. McLuckey, J. Wu, J.L. Bundy, J.L. Stephenson, G.B. Hurst, Oligonucleotide mixture analysis via electrospray and ion/ion reactions in a quadrupole ion trap, Anal. Chem. 74 (2002) 976–984. https://doi.org/10.1021/ac011015y.

[9] J. Stephenson, S.A. McLuckey, Ion/Ion reactions for oligopeptide mixture analysis: Application to mixtures comprised of 0.5-100 kDa components, J. Am. Soc. Mass Spectrom. 9 (1998) 585–596. https://doi.org/10.1016/S1044-0305(98)00025-7.

[10] S.A. McLuckey, J.L. Stephenson, Ion/ion chemistry of high-mass multiply charged ions, Mass Spectrom. Rev. 17 (1998) 369–407. https://doi.org/10.1002/(SICI)1098-2787(1998)17:6<369::AID-MAS1>3.0.CO;2-J.

[11] K.J. Laszlo, M.F. Bush, Analysis of Native-Like Proteins and Protein Complexes Using Cation to Anion Proton Transfer Reactions (CAPTR), J. Am. Soc. Mass Spectrom. 26 (2015) 2152–2161. https://doi.org/10.1007/s13361-015-1245-4.

[12] L. Ding, M. Sudakov, S. Kumashiro, A simulation study of the digital ion trap mass spectrometer, Int. J. Mass Spectrom. 221 (2002) 117–138. https://doi.org/10.1016/S1387-3806(02)00921-1.

[13] N.M. Hoffman, Z.P. Gotlib, B. Opačić, A.P. Huntley, A.M. Moon, K.E.G. Donahoe, G.F. Brabeck, P.T.A. Reilly, Digital Waveform Technology and the Next Generation of Mass Spectrometers, J. Am. Soc. Mass Spectrom. 29 (2018) 331–341. https://doi.org/10.1007/s13361-017-1807-8.

[14] K.W. Lee, G.S. Eakins, M.S. Carlsen, S.A. McLuckey, Increasing the Upper Mass/Charge Limit of a Quadrupole Ion Trap for Ion/Ion Reaction Product Analysis via Waveform Switching, J. Am. Soc. Mass Spectrom. 30 (2019) 1126–1132. https://doi.org/10.1007/s13361-019-02156-z.

[15] K.W. Lee, G.S. Eakins, M.S. Carlsen, S.A. McLuckey, Ion trap operational modes for ion/ion reactions yielding high mass-to-charge product ions, Int. J. Mass Spectrom. 451 (2020) 116313. https://doi.org/10.1016/j.ijms.2020.116313.

[16] J.M. Wells, P.A. Chrisman, S.A. McLuckey, Formation and characterization of protein-protein complexes in vacuo, J. Am. Chem. Soc. 125 (2003) 7238–7249. https://doi.org/10.1021/ja035051l.

[17] H.-C. Chao, M. Shih, S.A. Mcluckey, Generation of Multiply Charged Protein Anions from Multiply Charged Protein Cations via Gas-Phase Ion/Ion Reactions, J. Am. Soc. Mass Spectrom. 31 (2020) 56. https://doi.org/10.1021/jasms.0c00062.

[18] F.L. Brancia, B. McCullough, A. Entwistle, J.G. Grossmann, L. Ding, Digital asymmetric waveform isolation (DAWI) in a digital linear ion trap, J. Am. Soc. Mass Spectrom. 21 (2010) 1530–1533. https://doi.org/10.1016/j.jasms.2010.05.003.

[19] G.F. Brabeck, P.T.A. Reilly, Mapping ion stability in digitally driven ion traps and guides, Int. J. Mass Spectrom. 364 (2014) 1–8. https://doi.org/10.1016/j.ijms.2014.03.008.

[20] P.E. Miller, M.B. Denton, The quadrupole mass filter: Basic operating concepts, J. Chem. Educ. 63 (1986) 617–622. https://doi.org/10.1021/ed063p617.

[21] J.N. Louris, J.S. Brodbelt-Lustig, R. Graham Cooks, G.L. Glish, G.J. van Berkel, S.A. McLuckey, Ion isolation and sequential stages of mass spectrometry in a quadrupole ion trap mass spectrometer, Int. J. Mass Spectrom. Ion Process. 96 (1990) 117–137. https://doi.org/10.1016/0168-1176(90)87025-C.

[22] P.H. Dawson, Fringing fields in the quadrupole mass filter, Int. J. Mass Spectrom. Ion Phys. 6 (1971) 33–44. https://doi.org/10.1016/0020-7381(71)83002-4.

[23] M. Van De Waterbeemd, K.L. Fort, D. Boll, M. Reinhardt-Szyba, A. Routh, A. Makarov, A.J.R. Heck, High-fidelity mass analysis unveils heterogeneity in intact ribosomal particles, Nat. Methods. 14 (2017) 283–286. https://doi.org/10.1038/nmeth.4147.

CHAPTER 6. CUSTOM-BUILT INSTRUMENT CONTROLLER

6.1 Introduction

In-house built instrumentation provides extra flexibility and control in experimental design; however, these custom experiments tend to rely on a complicated setup of different electronics. This chapter describes a custom instrument controller that facilitated several experiments using digital ion trap technology. The controller has five main sections: (1) ShieldBuddy microcontroller that reads and executes instructions from the computer software, (2) waveform generation, (3) digital outputs, (4) analog outputs, and (5) Arduino Due microcontroller that digitizes raw data and sends the data bytes to the computer. Both the ShieldBuddy and Arduino Due were programmed using the Arduino integrated development environment (IDE) with built-in functions for Arduino boards. The custom software was programmed using PyQt5 [1] and several Python packages including NumPy [2], PyQtGraph [3], Matplotlib [4], and PySerial [5].

6.2 Instrument Controller Architecture

6.2.1 ShieldBuddy Microcontroller

The instrument controller is contained to one printed circuit board and powered by an external power supply with leads for +15 V, ground, and −15 V, shown in Figure 6.1. A ShieldBuddy microcontroller receives scan function instructions from the computer, creates the scan function in its memory, and executes the scan function segments sequentially by updating output voltages and frequencies of the other controller components. The code used to program the ShieldBuddy is in Appendix A, and a schematic of the ShieldBuddy is in Figure A.1.

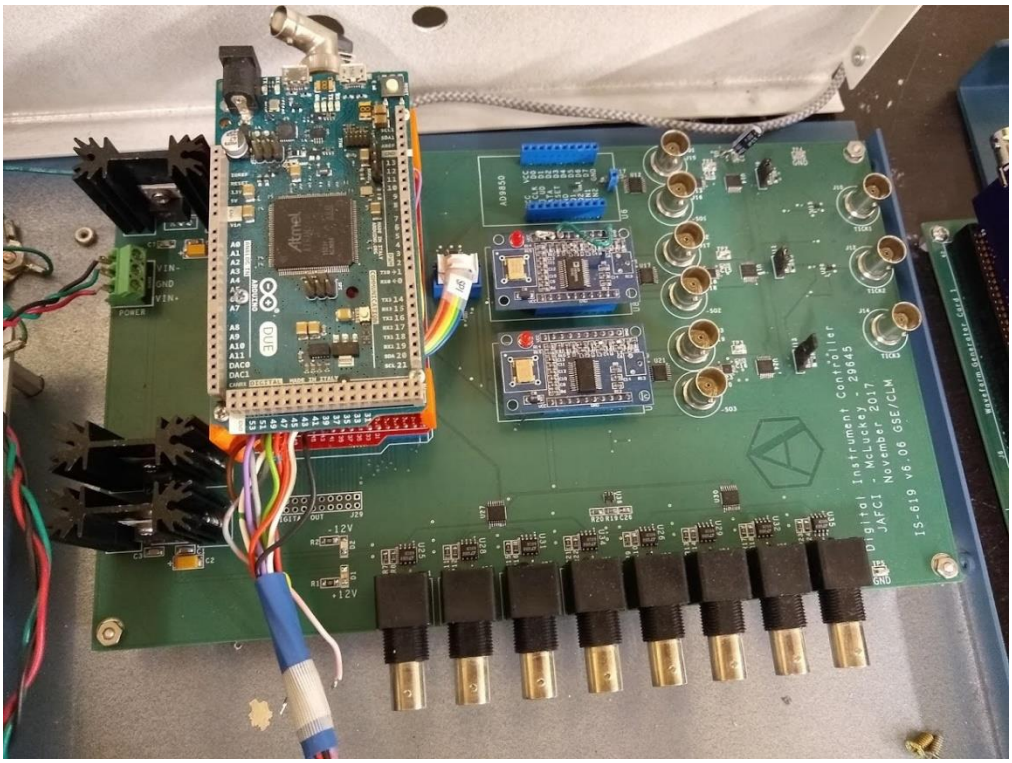

Figure 6.1. Instrument controller board. Red ShieldBuddy microcontroller (mostly hidden in upper left) receives scan function information from computer through USB. The ShieldBuddy instructs three waveform generator cards using SPI communication (upper middle with one missing) to produce three square wave outputs, produces 12 digital ouptuts for external triggering (wires coming from Shieldbuddy in lower left), and controls eight analog outputs through two DACs using I^2C communication (lower portion of main board). It also triggers an Arduino Due (mounted above the Shieldbuddy in upper left corner) which collects real-time data and sends it to the computer through a second USB port.

6.2.2 Waveform Generation

The controller is equipped with three waveform generator cards (AD9850 CMOS complete direct digital synthesizer, Analog Devices). The AD9850 uses direct digital synthesis (DDS) for rapid and accurate frequency changes necessary in high resolution frequency scans in digital ion trap (DIT) mass analysis [6], and can be controlled using the ShieldBuddy's serial peripheral interface (SPI). The Arduino IDE has built-in functions for controlling SPI communication; however, communication with the AD9850s was accelerated by manually controlling the SPI pins of the ShieldBuddy. Each AD9850 generates two phases of a 0–5 V square wave (BNC connectors directly to right of the AD9850s in Figure 6.1), and one phase is directed through a clock divider (SN74LV163APWR 4-bit synchronous binary counter, Texas Instruments) to generate a second square wave that is phase-locked to the original but with a frequency that is either one-half, one-quarter, one-eighth, or one-sixteenth of the original frequency. This frequency-divided square

wave is used as the dipolar excitation waveform for mass analysis. Duty cycles for the three original square waves and amplitudes for the frequency-divided square waves are controlled by digital potentiometers (MCP4661 dual I^2C digital POT, Microchip). Two ShieldBuddy pins were configured with built-in Arduino functions for inter-integrated circuit (I^2C) communication with the MCP4661 potentiometers. Figures A.3 and A.4 show schematics of the waveform generation circuitry.

6.2.3 Digital Outputs

To create digital outputs used for triggering scan function events (e.g., ionization events, external waveform generators, high voltage pulse generators) digital pins 36–53 were connected to BNC connectors with Schottky diodes as protection from voltage or current surges. The ShieldBuddy board has extra Arduino functions for faster updates of the digital pins than are achieved by other Arduino boards. This means that event triggering is not completely synchronous as desired, but rather there is a 30–40 ns delay between neighboring events. A schematic of the digital outputs is in Figure A.1.

6.2.4 Analog Outputs

There are two 4-channel digital to analog converters (DAC8574, Texas Instruments), which control a total of eight analog outputs. Analog outputs are used to control external supplies that take a low voltage input to generate a high voltage output. The ShieldBuddy communicates with the DACs using a second set of I^2C pins. Unlike the digital outputs, the analog outputs can be updated synchronously by first sequentially loading values into the eight channels of the two DACs and then sending a master update signal to both DACs. Because the two DACs share the same I^2C pins, synchronous updating is possible. Figure A.5 is the schematic of the analog output circuitry.

6.2.5 Data Collection with Arduino Due Microcontroller

The Arduino Due was programmed to sample the voltage output of the instrument detector at a rate of one mega sample per second. It waits for a signal from digital pin 13 on the ShieldBuddy to trigger data sampling. As data are sampled, they are sent via USB to the computer

for real-time acquisition and plotting. The code used to program the Arduino Due is in Appendix A.

6.3 Instrument Control Software

All Python code written for the program is in Appendix B. When starting the software, two windows appear: the main control window titled "Scan Function Creator" shown in Figure 6.2 and the secondary data window titled "Data Collector and Viewer" shown in Figure 6.7. The program can only be closed from the main window (clicking on the X in the data window will not close the program).

6.3.1 Main Window

Below the menus taking up most of the window in Figure 6.2 is the scan function definition area. Segments can be added, removed, and edited before sending the scan function to the controller. On the right is an area with four buttons that controls communication with the controller and an area that prints communication received back from the controller. A scan function segment has six sections. In the top section, the user can name the section (for reference only, has no functionality), choose whether the segment will be active when running the scan function, choose whether data is recorded during that segment, and define the segment length in milliseconds (ms). The "Active" and "Record" variables are set by typing a "1" for "True" and a "0" for "False". The second, third, and fourth sections define the square wave outputs of the instrument controller. Each square wave output has a start and end frequency and duty cycle that defines the trapping waveform. The "Tickle" variable defines what kind of square wave will be used for dipolar excitation. "Output 3" (the default entry) means that the square wave defined as the third waveform output will be used for excitation. This operation is useful for applications that target a specific *m/z* at a specific secular frequency, like collision-induced dissociation (CID). Other valid entries for the "Tickle" variable are 2, 4, 8, and 16, which define a dipolar excitation frequency ramp by dividing the frequencies used in the trapping square wave ramp by 2, 4, 8, or 16, respectively. This operation is useful for mass analysis, because the trapping and excitation frequencies are ramped at a constant division factor to ensure that ions are ejected at a constant q value during the frequency ramp. "Amplitude" and "Phase" define the amplitude (0 to 5 V) and

relative phase offset (0° or 180°) of the dipolar excitation square wave. The fourth and fifth sections define analog and digital output states during the scan function segment. Analog outputs can range from −10 to 10 V, and digital outputs can either be "True" (by typing a "1") or "False" (by typing a "0"). Square wave duty cycle, excitation amplitude, and analog outputs are all uncalibrated. For example, to get a 50% duty cycle output, the user will likely need to type in a "58". To produce a 0 V output on the analog outputs, the user will likely need to type in about "−0.55". These can be calibrated by editing the Shieldbuddy Arduino code that interprets the scan function. In any case, it is recommended to view all outputs on an oscilloscope prior to running an experiment.

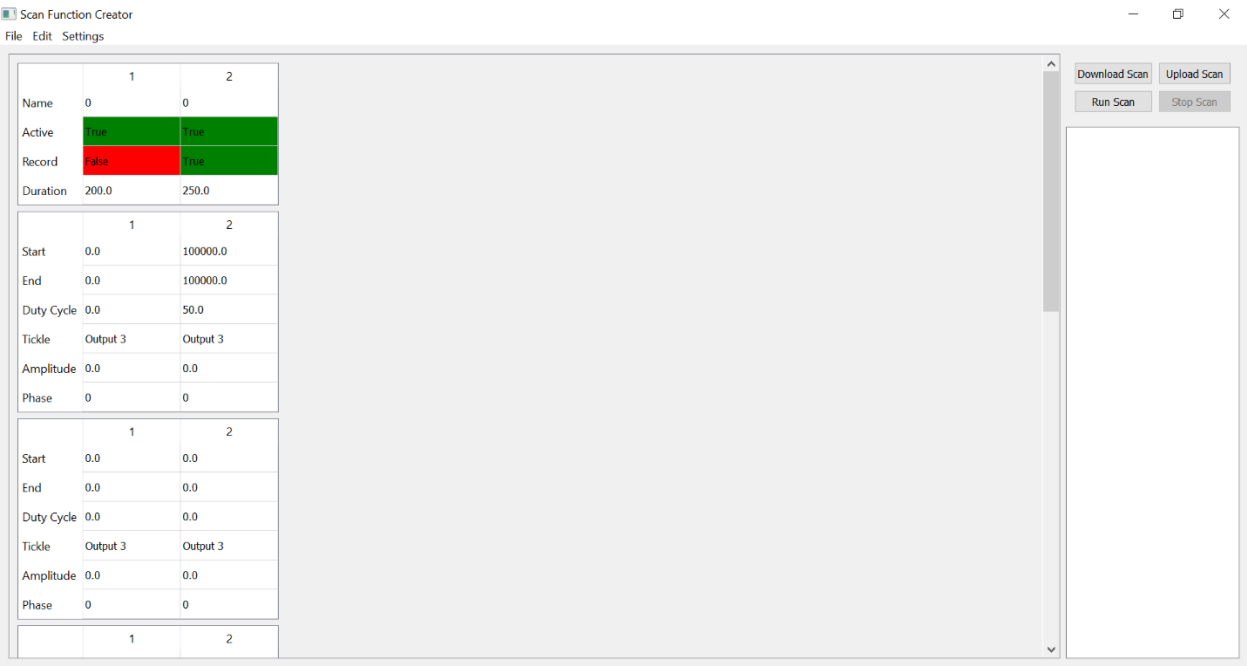

Figure 6.2. Main window of instrument control software. Central area contains editable tables to define a scan function. Right area contains buttons to communicate with the instrument controller and a text box that prints communications received from the controller.

The File menu has options to save a scan function to file or open a previously saved scan function. Saving a scan function will create a text file with a *.scan file extension for recognizability and a jpeg file that is a screenshot of the scan function definition as a user-friendly reference.

The Edit menu has two working options. "Add/Remove segments" will open a small dialog box shown in Figure 6.3. A new segment can be added by specifying before which existing

segment the new segment should be placed and clicking the “Insert segment” button. An existing segment can be removed by specifying the segment number and clicking the “Remove segment” button.

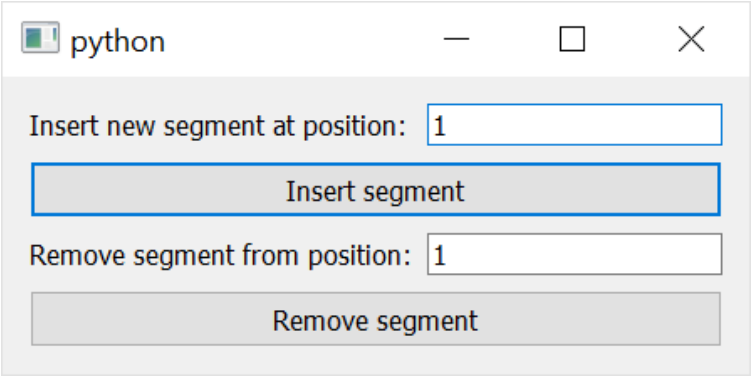

Figure 6.3. Dialog window for adding and removing segments to a scan function.

The “Calculator” Edit menu option opens a dialog box shown in Figure 6.4. The user can enter the radial and axial field dimensions for a 3D ion trap, the high and low voltages used in the square wave, the duty cycle of the square wave, the square wave frequency, and ion *m/z* value. The calculator will then calculate the radial and axial beta values and omega (ion secular frequency) values. If a target beta value is entered, either the “Frequency” or “*m/z*” button can be clicked which will adjust the frequency or *m/z*, respectively, until the calculated axial beta value matches the user-entered target beta value. This action also calculates a constant that relates frequency to *m/z* for a resonance ejection scan at the specified target beta value and loads that constant into the data window for crude mass calibration of spectra. Clicking the “Plot” button will generate a stability region with frequency on the *x* axis and duty cycle on the *y* axis. The yellow region designates axial stability, the blue designates radial stability, and the pink region is the overlap. A black circle designates the position of the entered *m/z* at the entered duty cycle and frequency. If any numbers are changed, the “Plot” button must be clicked again to recalculate the position of the *m/z* of interest.

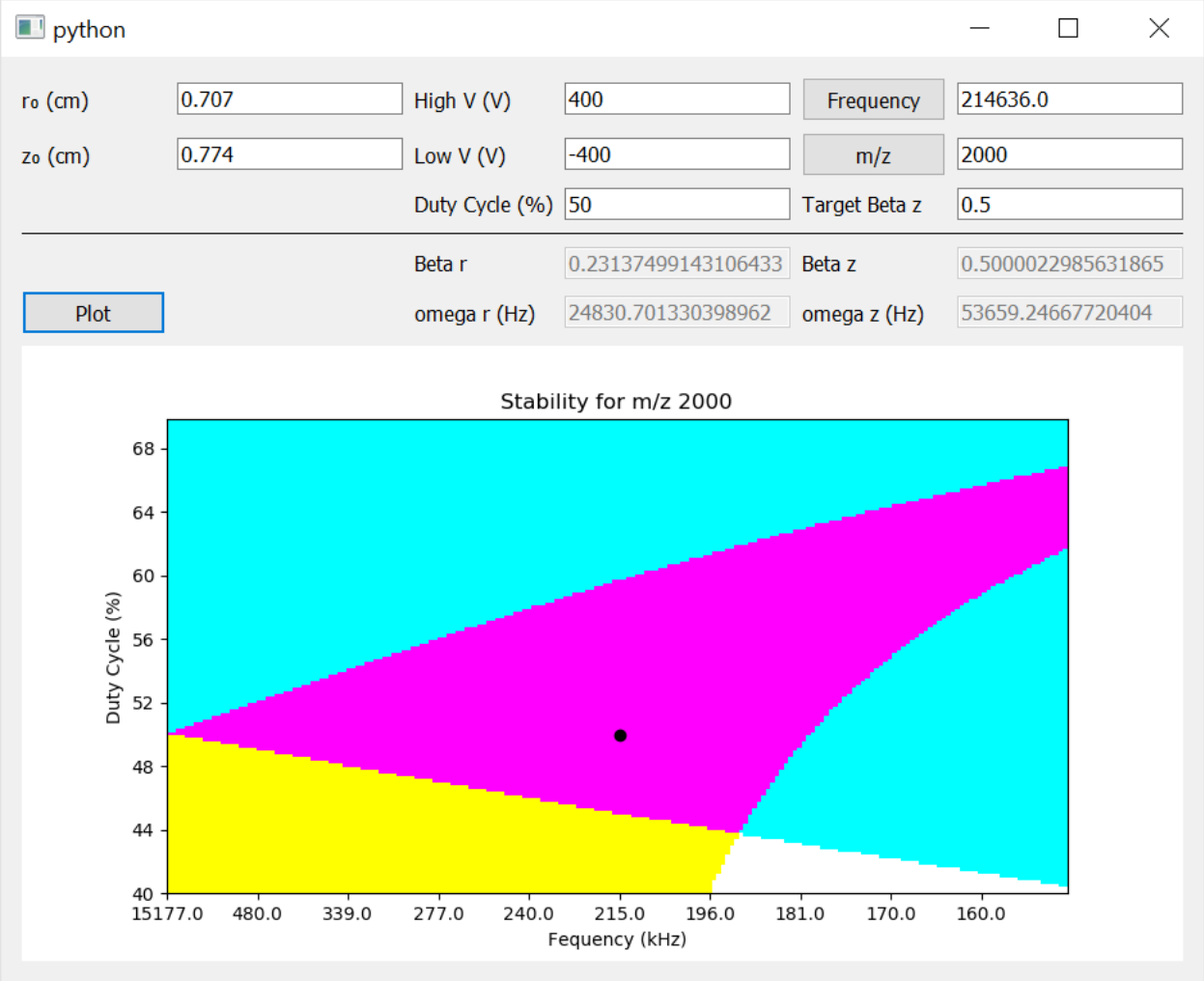

Figure 6.4. Calculator dialog window designed for square wave calculations only. The Frequency button calculates the frequency needed to put the entered *m/z* value at the target beta value. The *m/z* button calculates the *m/z* that will be at the target beta value given the entered frequency. The Plot button calculates and plots a stability region with a black circle designating the entered *m/z*. Yellow is axial stability, blue is radial, and pink is the overlap.

The Settings menu has an option to establish connections with the instrument controller and another option to adjust data collection. Choosing the "Connect" option opens a dialog window shown in Figure 6.5. Because the instrument controller relies on two USB connections (one for the ShieldBuddy microcontroller and one for the Arduino Due microcontroller), the dialog window has areas for defining two USB COM ports. The ShieldBuddy is connected via the Control COM port, and the Arduino Due is connected via the Data COM port. Clicking either the "Connect Controller" or "Connect Data" button will cause the software to attempt to establish a USB connection through the specified port. Clicking the either disconnect button will cause the

software to release its connection to the specified USB port. In all cases, the right communication area of the main window will print whether a connect action or disconnect action was successful.

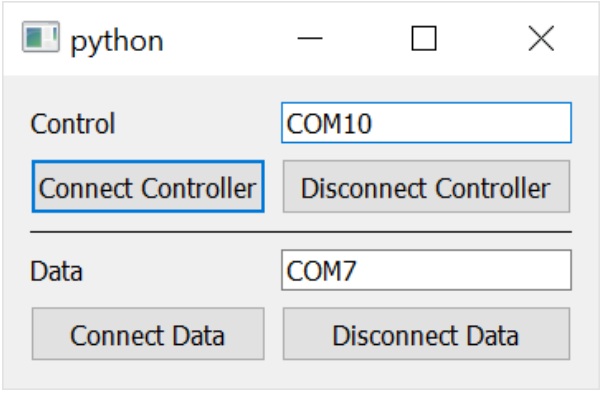

Figure 6.5. Dialog window for establishing connections with the instrument controller. The COM port for the ShieldBuddy USB connection is typed next to "Control" and the COM port for the Arduino Due USB is typed next to "Data".

Choosing the "Data Settings" option from the Settings menu opens a dialog window shown in Figure 6.6. Currently, there is only one setting to adjust. Real-time data is collected at a constant rate of 1 mega sample per second (MSPS), or one million 8-bit values per second. (Note that each data point is actually 16 bits, so the data rate is 0.5 MSPS.) Storing one million 8-bit numbers requires 1 megabyte (MB) of memory. Thus, averaging 100 scans that are one second long would require 100 MB, which can start to be demanding on a computer. (For reference, many large programs use 300–600 MB.) Additionally, it requires extra computer resources and time to average a large number of scans together. To allow users to take long scans and/or average a large number of scans without using too much computer memory, the data point sampling rate can be specified. The number entered tells the program to keep one data point for every *x* data points, where *x* is the number entered (e.g., entering 4 tells the program to keep one data point and throw out the next three).

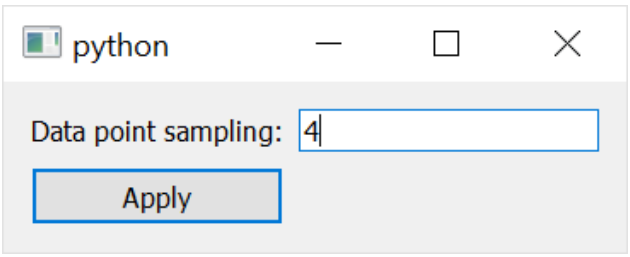

Figure 6.6. Data settings dialog window. Currently, the only setting to adjust is the number of data points to down sample by when collecting and plotting real-time data.

6.3.2 Data Window

The data window has three plot areas. The top shows real-time spectra as they are collected during an experiment. Changing the number by averages will cause the program to hold that number of spectra in computer memory and calculate an average spectrum from them. The average spectrum in the plot will be continuously updated until the specified number of spectra are collected. At this point, the program will dump the oldest spectrum from memory and replace it with the newest collected spectrum and repeat this process until the scan function is stopped or the averaging number is changed. Clicking the Save button will save the data points of the averaged spectrum to a text file. Typing in the start and end frequencies and clicking the Calibrate button will give a crudely calibrated *m/z* scale on the x axis based on the displayed constant. From the Mathieu stability parameter equations, $\Omega^2 q_{eject}(m/z) = constant$ if the trapping waveform amplitude is constant. In digital frequency scanning experiments, the amplitude is constant and ions are ejected at a constant q_{eject}; therefore, the *m/z* range can be calculated from the frequency range (assuming a linear *m/z* scan) using a constant specific to the experiment. The user can manually change this constant, or the built-in calculator – accessed from the Edit menu of the main window – will automatically update it based on entered parameters. Clicking the "Calibrate" button will calculate and display a new x axis based on the entered frequencies and constant. If the calibration fails for any reason, the default x axis will be displayed.

The middle plot area is used for viewing previously saved spectra. The user can open a previously saved file using the "Open" button and choosing the text file containing the saved data. Technically any text containing two columns of tab delimited numbers can be opened and plotted. This plot area can be individually mass calibrated using the same procedure as the top plot area.

The bottom plot area displays a total intensity vs. time plot for the collected real-time scans. This mimics a total ion count (TIC) plot that commercial mass spectrometer software typical has;

however, the total intensity is not directly comparable to the number ions. Rather, each point on the plot is a raw sum of all the data in one real-time spectrum. The x axis simply counts up by one rather than reflecting an actual time scale.

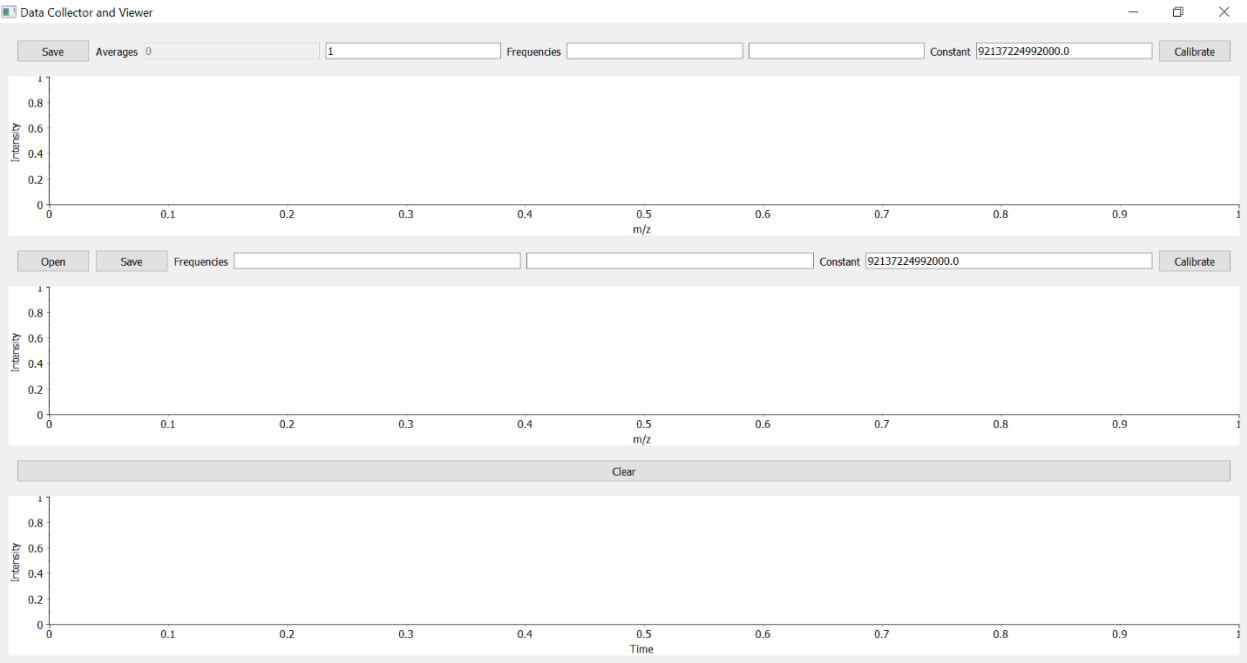

Figure 6.7. Data window of control software. Real-time data is processed and plotted in the top plot. Saved data can be loaded and plotted in the middle plot for viewing. The bottom plot is a total ion count (TIC) plot made by plotting the total intensity from each real-time spectrum in the top plot against time.

6.4 First-Time Setup for the Instrument Controller

6.4.1 Requirements for Using the ShieldBuddy Microcontroller

The ShieldBuddy microcontroller (Hitex Ltd., UK) is a tri-core microcontroller with the form of an Arduino™ Due. For those familiar with Arduino programming, the ShieldBuddy can be programmed and operated in the same way after installing some tools. The software tools require Windows Vista or later. Download and install the HighTec Free TriCore™ Entry Tool Chain from https://free-entry-toolchain.hightec-rt.com/. Fill out the form to receive a free license for the computer which will be programming and running the ShieldBuddy. The license expires every so often, so the license will need to be redownloaded on occasion. The license file should be placed in the C:\HIGHTEC\licenses folder on the computer's hard drive. Download the latest version of the Arduino software by visiting http://arduino.cc/download.php?f=/arduino-1.6.13-

windows.exe. Visiting this link should automatically trigger the download. Download the Arduino development environment add-in from http://www.hitex.co.uk/fileadmin/uk-files/downloads/ShieldBuddy/ShieldBuddyMulticoreIDE.zip, which allows the standard Arduino software to recognize the ShieldBuddy controller. Unzip the file using the password "ShieldBuddy", and run the installer using the same password. At this point, run the Arduino software with the ShieldBuddy connected to a USB port to verify that the Arduino software recognizes the connected ShieldBuddy. For more information and help, refer to the user manual and Getting Started guide, which can be downloaded from http://www.hitex.co.uk/index.php?id=3650.

6.4.2 Arduino Code Modifications

A few modifications were made to an existing ShieldBuddy file to include a few capabilities that are used in the instrument controller code. Two copies of a file titled "Variant.h" are found in "C:\Program Files\Arduino\hardware\aurduino_Dx\aurix\variants\tc275\" and "C:\Program Files\Arduino\hardware\aurduino\aurix\variants\tc275\". The first file path (with the "aurduino_Dx" folder) is for ShieldBuddys with a DC-step processor (SAK-TC275TP-64F200N DC, Infineon). The second file path (with the "aurduino" folder) is for ShieldBuddys with a CA-step processor (SAK-TC277TP-64F200S CA, Infineon). Regardless, both Variant.h files can be changed with no negative consequences.

First, line 688 in Variant.h should be changed to:

```
#define D42_index     10
```

This line allows for fast access to the digital pins on the ShieldBuddy, which are used as the instrument controller digital outputs. The original file has an incorrect pin definition for pin D42. Additional pin definitions allow fast access to the pins used for SPI communication with the waveform generator circuits, which can significantly decrease the time necessary to update output frequencies. With faster frequency updates, the controller can produce higher resolution frequency scans (i.e., more frequency steps in the same amount of time), which can produce higher resolution mass spectra. Adding the following lines at line 742 in Variant.h will allow faster access to the communication pins:

```
#define D62_OMSR   P22_OMSR.U
#define D62_OMCR   P22_OMCR.U
#define D62_index     3

#define D63_OMSR   P22_OMSR.U
#define D63_OMCR   P22_OMCR.U
#define D63_index     0
```

These lines define the SPI clock pin as pin 62 and the SPI data pin as pin 63.

The current instrument controller uses pin 13 on the ShieldBuddy to trigger the Arduino for data acquisition. The next version of the controller positions the ShieldBuddy and Arduino Due on the main board in such a way that pin 13 is not easily accessible for producing this trigger. The simplest solution was to reconfigure an analog pin on the ShieldBuddy to trigger the Arduino Due. Adding another pin definition to Variant.h accomplishes this:

```
#define DA5_OMSR  P10_OMSR.U
#define DA5_OMCR  P10_OMCR.U
#define DA5_index    7
```

The above lines define pin A5 on the ShieldBuddy as a fast access digital pin. Normally, a pin is defined as a digital output pin using the built-in Arduino function "pinMode(##, OUTPUT)" where ## represents the pin number. Because the analog pins are not intended for digital output, a hardware definition needs to be added to the Arduino code for the ShieldBuddy. Adding the following code to the Arduino "setup()" function defines pin A5 as a digital output pin:

```
IfxPort_setPinModeOutput(&MODULE_P10, 7, IfxPort_OutputMode_pushPull,
                                    IfxPort_OutputIdx_general);
```

Pin A5 can then be accessed using "Fast_digitalWrite(A5, LOW or HIGH)" just like the normal digital pins, which allows it to trigger the Arduino Due for data acquisition.

6.4.3 Requirements for Running the Instrument Control Software

The custom software was designed using PyQt5 (Riverbank Computing, UK), which is a Python binding for the Qt framework. Qt is a tool for designing and building software user interfaces. The code for the Instrument Controller is hosted at https://github.com/klee200/DITController, where it can be cloned or downloaded. To run the main python script, several Python packages need to be installed. Download and install Python 3.7 from https://www.python.org/downloads/. The version number is important, because the packages used work correctly with Python 3.7 but not 3.8, for example. As older versions of Python and the various packages become obsolete, it might be necessary to update the versions and likely some of the code itself. This will have to be done ad hoc by debugging the various files that build the software. After Python is installed, open a Command Prompt window, and run the following:

```
python -m pip install -U pip
```

which will install and update PIP, a Python package manager. Navigate to the folder with the instrument controller software by typing:

```
cd C:\[insert path to folder here]
```

Install the required packages by running the command:

```
pip install -r requirements.txt
```

which will tell PIP to read the required packages from the provided requirements.txt file and install them. To run the program, type the command:

```
python main.py
```

which will run the code in the main.py file as a Python script. If the program fails to run, read the error information in the command prompt window for help with debugging.

6.5 References

[1] M. Summerfield, Rapid Gui Programming with Python and Qt: The Definite Guid to PyQt Programming, Prentice Hall, Upper Saddle River, NJ, 2008.

[2] T.E. Oliphant, A guide to NumPy, Trelgol Publishing, USA, 2006.

[3] L. Campagnola, PyQtGraph, (2020). https://github.com/pyqtgraph/pyqtgraph.

[4] J.D. Hunter, Matplotlib: A 2D graphics environment, Comput. Sci. Eng. 9 (2007) 90–95. https://doi.org/10.1109/MCSE.2007.55.

[5] C. Liechti, PySerial, (2017). https://github.com/pyserial/.

[6] H. Koizumi, B. Jatko, W.H. Andrews, W.B. Whitten, P.T.A. Reilly, A novel phase-coherent programmable clock for high-precision arbitrary waveform generation applied to digital ion trap mass spectrometry, Int. J. Mass Spectrom. 292 (2010) 23–31. https://doi.org/10.1016/j.ijms.2010.02.011.

CHAPTER 7. CALCULATION-BASED SPECTRAL PREDICTIONS

7.1 Introduction

Mass spectra inherently represent multidimensional data because the x-axis is related to mass and charge. In the case of polymers and multimers, a third variable hidden in the mass-to-ratio is the number of monomers or units. Mass spectral deconvolution is a probability-based approach to determine the mass and charge of individual peaks [1–3]. Because this in effect increases the dimensionality of the data (i.e., one-dimensional mass-to-charge to two-dimensional mass vs. charge), deconvolution requires an assumption about the relationship among peaks. Typical approaches assume that neighboring peaks differ in mass by the charge carrying unit (e.g., a proton) and differ by one charge. Deconvolution will then group a distribution of neighboring charge states as having the same identity. Other factors such as peak widths due to desolvation, salt adduct distributions, and the presence of other analytes with their own charge state distributions can create many artifacts in the deconvolution because peaks can be incorrectly grouped.

Ion/ion reactions can facilitate deconvolutions by increasing confidence in charge state assignments [4–6]. For example, proton transfer reactions generate well resolved low charge products that differ in mass and charge by single protons. Spectra of high mass and/or highly adducted ions do not typically benefit from single proton transfers; however, reactions with larger and more highly charged ions can lead to multiple proton transfers from a single collision or an attachment of the reagent ion [7,8]. In either case, the product peaks are well separated from the precursor analyte peaks with a known difference in mass and charge.

Even with assistance from ion/ion reactions, deconvolutions are still prone to artifacts and errors, especially when measuring heterogeneous analytes [9]. One way to distinguish between true peaks in the deconvolution and artifacts is to independently verify them by effectively "reconvolving" the determined masses and charges and comparing the result with the obtained data. The process described here of generating a spectrum has two major pieces: (1) Calculating parameters for individual peaks based on the distributions of masses and charges and (2) combining the individual peaks into a spectrum to plot on a common set of axes. The approach presented here defines each peak by its position (or *m/z* value), intensity, and gaussian peak width.

7.2 Calculations

7.2.1 Calculating Peak Parameters

The convolution uses the same basic assumptions used in the deconvolution, that is, that neighboring charge states in a single distribution differ in mass and charge by the charge carrier (usually a proton).[1] Determining peak positions is then simply:

$$(m/z)_j = \frac{m + jm_c}{jz_c} = \frac{m}{jz_c} + (m/z)_c \tag{7.1}$$

where *j* is the number of charge carriers condensed on the analyte, *m* is the analyte mass, m_c is the charge carrier's mass, and z_c is the charge carrier's charge. Anions will return a negative peak position; therefore the absolute value of *m/z* can be used for plotting purposes. Including a mass distribution, such as an istopic distribution, gives:

$$(m/z)_{i,j} = \frac{m_i + jm_c}{jz_c} = \frac{m_i}{jz_c} + (m/z)_c \tag{7.2}$$

where m_i is the mass of the *i*th component in the mass distribution.

To expand this model to a polymer with repeating units of equal mass requires introducing a size distribution. A general representation includes unique mass distribution for a non-repeating unit and a repeating unit. The size distribution describes the number of repeating units attached to the non-repeating unit and is used to calculate the total mass range of the polymer. Including a size distribution for mass gives:

$$(m/z)_{o,p,j,k} = \frac{m_o + km_p}{jz_c} + (m/z)_c \tag{7.3}$$

where *k* is the number of repeating units present in the observed mass-to-charge, m_o is the *o*th mass from the non-repeating mass distribution, and m_p is the *p*th mass from the repeating mass distribution. To determine the range of *j* for a polymer, the model requires a distribution that describes how many charge carriers per repeating unit will condense on the polymer. The peak positions are then given by:

$$(m/z)_{o,p,n,k} = \frac{m_o + km_p}{nkz_c} + (m/z)_c \tag{7.4}$$

where *n* is a chosen ratio of charge carriers per polymer size. This model creates fractional numbers of charge carriers. For example, to describe a polymer with a charge carrier on every other repeating unit of a polymer with length five would yield 2.5 charge carriers. Figure 7.1

illustrates this case for poly-lysine. After calculating the total fractional number of charge carriers, this number can be rounded to both whole number values that surround it to give:

$$(m/z)_{o,p,n,k} = \frac{m_o + km_p}{jz_c} + (m/z)_c, \quad j = \begin{cases} \lceil nk \rceil \\ \lfloor nk \rfloor \end{cases} \tag{7.5}$$

where *j* is the result of applying both a ceiling and floor operation to fractional values, giving two peaks for every fractional charge.

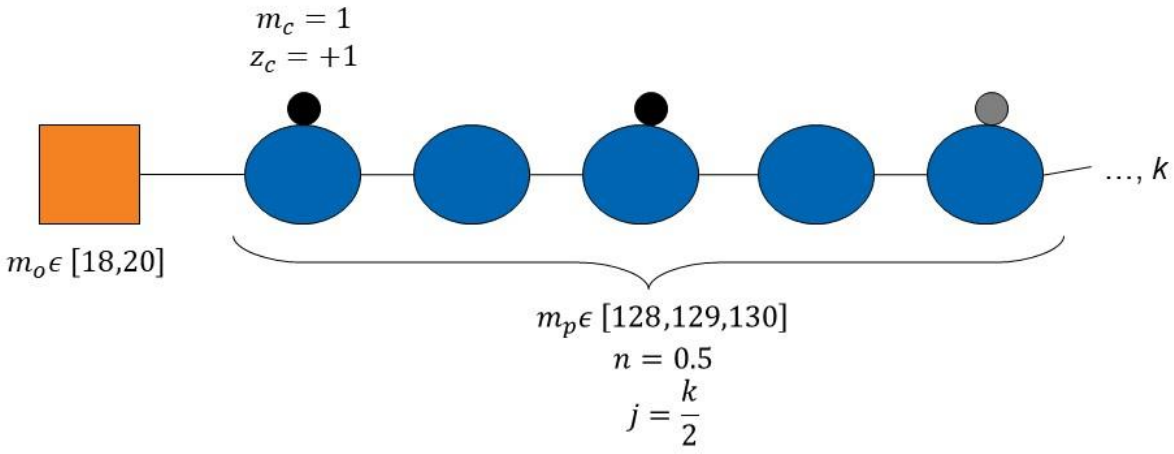

Figure 7.1. Graphical representation of polymer model using poly-lysine as an example. The non-repeating unit (orange) is the combined C-terminal OH and N-terminal H with a mass distribution including 18 and 20 Da. The repeating unit (blue) has a mass distribution including 128, 129, and 130 Da. This model predicts that a proton (black, with mass of 1 Da and charge of +1) will condense on every other repeating unit, thus $n = 0.5$. The gray proton indicates that with $k = 5$, j is rounded to 2 and 3.

This model also can describe atypical polymers. One example is the result of an ion/ion reaction between two opposite polarity proteins. The reaction will result in several different gas-phase complexes, such as the cationic protein being bound to zero, one, two, or three anionic proteins. Although the anionic protein was not originally a polymer, it mathematically appears as a polymer of sizes zero to three. In this case, m_0 and z_0 are both zero whereas m_p and z_p are the mass and charge of the anionic protein.

A simple method for calculating peak intensities is to assume an intensity distribution for each distribution contributing to the peak position (e.g., polymer size distribution, charge state distribution, etc.) and multiply the intensity distributions together. Assuming that charge state distributions and polymer size distributions are gaussian would give:

$$h_{o,p,n,k} = f_o(m_o) \times f_p(m_p) \times f_n(n) \times f_k(k) \tag{7.6}$$

where f_o and f_p are the mass distribution intensities of the non-repeating and repeating units, respectively. The number of charge carriers per polymer unit and polymer size each have their own intensity distributions represented by f_n and f_k, respectively. Whereas the mass distribution intensities should be calculated from isotopic abundances, the distribution intensities of charge carrier number and polymer size are more readily modelled by gaussian, poisson, or gamma distributions.

Plotting the calculated *m/z* vs. *h* values will only show peak centroid positions, sometimes referred to as a "stick spectrum". This is appropriate when modeling isotopically resolved spectra; however, it is not computationally feasible to calculate and plot isotopic distributions of large analytes. High mass analytes are more reasonably represented as a parameterized distribution, such as a gaussian, rather than a discrete distribution with intensities for every possible mass. For a polymer with a high mass non-repeating unit and high mass repeating units, the mass distribution for each size in the size distribution is a convolution of the non-repeating and repeating mass distributions.

$$f(m_{o,p,k}) = f_o(m_o) \otimes [f_p(m_p) \otimes f_p(m_p) \otimes \ldots]_k \tag{7.7}$$

For gaussian mass distributions, this is simplified by the fact that a convolution of two or more gaussians gives another gaussian whose variance is the sum of the starting gaussians' variances. The width of each mass distribution would be:

$$w_{o,p,k} = \sqrt{w_o^2 + k w_p^2} \tag{7.8}$$

where w_o is the mass distribution width of the non-repeating unit and w_p is the mass distribution width of the repeating unit. These mass distributions do not correspond to the peak widths of the mass spectrum, because charge will proportionally decrease these widths. Thus, the full peak width calculation given gaussian mass distributions (assuming negligible contribution from the mass distribution of the charge carrier) is:

$$w_{o,p,n,k} = \frac{\sqrt{w_o^2 + k w_p^2}}{j z_c}, \quad j = \begin{cases} \lceil nk \rceil \\ \lfloor nk \rfloor \end{cases} \tag{7.9}$$

This calculation applies to any peak width measurement (gaussian standard deviation, full-width half-max, etc.) because they all differ by constants. One step to further model real spectra is to include instrumental resolution. Equation (7.9 simply becomes:

$$w_{o,p,n,k} = \max\left(\frac{\sqrt{w_o^2 + kw_p^2}}{jz_c}, w_r\right), \quad j = \begin{cases}\lceil nk \rceil \\ \lfloor nk \rfloor\end{cases} \tag{7.10}$$

where w_r is the instrument resolution converted to the same units as the peak width calculation for comparison. As an important side note, this model suggests that instrumental resolution becomes less important as the unresolved analyte mass distribution width increases.

Using the three defined parameters – position, intensity, and width – a gaussian can represent each peak defined by:

$$\overrightarrow{y_{peak}} = h_{peak} \times \exp\left(-\frac{\left[\overrightarrow{x_{peak}} - (m/z)_{peak}\right]^2}{w_{peak}^2}\right) \tag{7.11}$$

where the subscript *peak* represents a combination of *o*, *p*, *n*, and *k* for a specific peak. The vectors x_{peak} and y_{peak} are plotted against each other to visualize the peak. Smoother, more defined peaks require more calculated (*x*, *y*) pairs at the cost of more computational resources.

7.2.2 Calculating Peak Parameters for Ion/Ion Reaction Products

After calculating peak positions, intensities, and widths for the cation and anion, the same three parameters can be calculated for all possible cation-anion products. A reaction product, or gas-phase complex, will have mass and charge equivalent to the sum of the cation and anion masses and charges, respectively, giving a peak position of:

$$(m/z)_{cat,an} = \frac{m_{cat} + j_{cat}m_{cat,c} + m_{an} + j_{an}m_{an,c}}{j_{cat}z_{cat,c} + j_{an}z_{an,c}} \tag{7.12}$$

where *cat* refers to the cation parameters and *an* refers to the anion parameters. For a common charge carrier (e.g., $m_{cat,c} = -m_{an,c}$ and $z_{cat,c} = -z_{an,c}$ such as with protonated and deprotonated molecules), the equation simplifies to:

$$(m/z)_{cat,an} = \frac{m_{cat} + m_{an}}{(j_{cat} + j_{an})z_{cat,c}} + (m/z)_{cat,c} \tag{7.13}$$

This calculation should be addressed carefully for isotopically resolved spectra. The gas-phase complexes formed by reactions between protonated, sodiated, potassiated, etc. molecules and deprotonated, chloridated, bromidated, etc. molecules will all have different *m/z* values because of the different masses of the charge carriers associated with the cations and anions.

Because peak intensities are the product of underlying distribution intensities, reaction product intensities are simply the product of the cation and anion intensities.

$$h_{cat,an} = h_{cat} \times h_{an} \tag{7.14}$$

Calculating reaction product peak widths is a similar approach as calculating product peak positions because of the different contributions from mass and charge. Extending the single peak width equation to two peaks (again ignoring peak width contribution from the charge carriers) gives:

$$w_{cat,an} = \max\left(\frac{\sqrt{w_{cat}^2 + w_{an}^2}}{z_{cat} + z_{an}}, w_r\right) \tag{7.15}$$

These parameters then describe the predicted gas-phase complexes produced by an ion/ion reaction using the same peak equation as before.

7.2.3 Combining Peaks into a Single Spectrum

Adding the individual peaks into a complete spectrum requires care, especially when peaks overlap (i.e., the individual x_i ranges overlap with each other). One solution is to use a common *x*-axis for all peaks and calculate *y* values for all peaks based on that common *x* axis before summing the individual peaks. As the number of peaks and/or distance between peaks increase, the computational requirements become increasingly difficult to satisfy. A more reasonable approach is to only calculate *y* values for peaks in the range of *x* where they are above a certain threshold. While a common *x*-axis still exists, individual peaks are only defined on small subsets of that common axis thereby decreasing the number of points stored in memory.

To illustrate how individual peaks can be combined into a single spectrum using this method, three hypothetical peaks will be used.

$$\begin{aligned}
\overrightarrow{y_1} &= 1 \times \exp\left(-\frac{[\overrightarrow{x_1} - 100]^2}{10^2}\right) \\
\overrightarrow{y_2} &= 2 \times \exp\left(-\frac{[\overrightarrow{x_2} - 200]^2}{5^2}\right) \\
\overrightarrow{y_3} &= 1 \times \exp\left(-\frac{[\overrightarrow{x_1} - 220]^2}{5^2}\right)
\end{aligned} \tag{7.16}$$

For gaussian peaks, four standard deviations from the mean gives a safe range to define the peaks giving ranges of:

$$\begin{aligned}\overrightarrow{x_1} &= [60, 140] \\ \overrightarrow{x_2} &= [180, 220] \\ \overrightarrow{x_3} &= [200, 240]\end{aligned} \tag{7.17}$$

A common step size for the *x*-axis is preferential but needs to be small enough to define all peaks. In this case, a step size of 10 will define peak 1 with nine points and peaks 2 and 3 with five points, not enough for good definition but enough for illustration purposes. The full ranges would then be:

$$\begin{aligned}\overrightarrow{x_1} &= [60 \quad 70 \quad 80 \quad 90 \quad 100 \quad 110 \quad 120 \quad 130 \quad 140] \\ \overrightarrow{x_2} &= [180 \quad 190 \quad 200 \quad 210 \quad 220] \\ \overrightarrow{x_3} &= [200 \quad 210 \quad 220 \quad 230 \quad 240]\end{aligned} \tag{7.18}$$

The *y* values are calculated from Equation (7.16) giving:

$$\begin{aligned}\overrightarrow{y_1} &= [0.00 \quad 0.01 \quad 0.14 \quad 0.61 \quad 1.00 \quad 0.61 \quad 0.14 \quad 0.01 \quad 0.00] \\ \overrightarrow{y_2} &= [0.00 \quad 0.27 \quad 2.00 \quad 0.27 \quad 0.00] \\ \overrightarrow{y_3} &= [0.00 \quad 0.14 \quad 1.00 \quad 0.14 \quad 0.00]\end{aligned} \tag{7.19}$$

The combined *x* range is the vector of unique *x* values from all peaks. Summing the *y* values first requires that the *x* ranges be aligned. For example:

$$\begin{bmatrix}\overrightarrow{x_1} \\ \overrightarrow{y_1} \\ \overrightarrow{x_2} \\ \overrightarrow{y_2} \\ \overrightarrow{x_3} \\ \overrightarrow{y_3}\end{bmatrix} = \begin{bmatrix} 60 & 70 & 80 & 90 & 100 & 110 & 120 & 130 & 140 & & & & & & & & \\ 0.00 & 0.01 & 0.14 & 0.61 & 1.00 & 0.61 & 0.14 & 0.01 & 0.00 & & & & & & & & \\ & & & & & & & & & 180 & 190 & 200 & 210 & 220 & & & \\ & & & & & & & & & 0.00 & 0.27 & 2.00 & 0.27 & 0.00 & & & \\ & & & & & & & & & & & 200 & 210 & 220 & 230 & 240 \\ & & & & & & & & & & & 0.00 & 0.14 & 1.00 & 0.14 & 0.00 \end{bmatrix} \tag{7.20}$$

The combined *x* and *y* would be:

$$\begin{bmatrix}\vec{x} \\ \vec{y}\end{bmatrix} = \begin{bmatrix} 60 & 70 & 80 & 90 & 100 & 110 & 120 & 130 & 140 & 180 & 190 & 200 & 210 & 220 & 230 & 240 \\ 0.00 & 0.01 & 0.14 & 0.61 & 1.00 & 0.61 & 0.14 & 0.01 & 0.00 & 0.00 & 0.27 & 2.00 & 0.41 & 1.00 & 0.14 & 0.00 \end{bmatrix} \tag{7.21}$$

Figure 7.2 is a plot of the total spectrum with points for the three individual peaks in different colors for comparison. Note that no memory was used to define the region between *x* = 140 and 180 where there were no peaks. A calculated spectrum can be compared point-by-point to a measured spectrum that is down sampled to the same range of *m/z* values as the calculated spectrum (i.e., the total *x* range). Alternatively, plotting the measured spectrum on the same set of axes as the raw measured spectrum can provide a visual comparison.

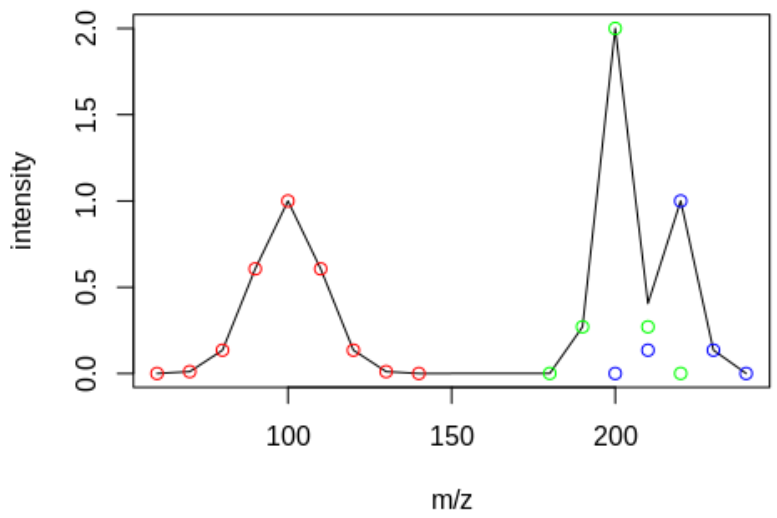

Figure 7.2. Plot of example of combining three individual peaks into one spectrum. Red, green, and blue points correspond to peaks 1, 2, and 3, respectively, from the above example. The black trace is the result of combining the peaks.

7.3 R Shiny App

The R language was designed for statistical computations and data analysis [10]. R Shiny [11] is a framework for creating simple user interfaces (UI) to execute scripts written in the R language. The app uses the following packages: rhandsontable [12], plotly [13], jsonlite [14], reshape2 [15], and dplyr [16]. In short, the app presents an interactive tab for defining an arbitrary number of positive and negative ions and calculates a positive mass spectrum, a negative mass spectrum, and a spectrum of the reaction complexes that could form. The various files used to build the app and perform the calculations are presented in Appendix C with inline comments.

The application allows a user to define separate isotopic distributions for the non-repeating and repeating parts of a polymer as well as the distribution of polymer lengths. To define a single ion rather than a polymer, the length is fixed as one. The distribution of polymer lengths is assumed to follow a gamma distribution parameterized using the mode (most probable value) and a loosely-defined "entropy" parameter. Gamma distributions are typically defined using a "shape" parameter and a "scale" parameter. The app uses "entropy" in place of "scale" to suggest that a higher entropy will lead to a wider distribution of polymer sizes. The shape parameter is calculated by $shape = \frac{mode}{entropy} + 1$. The user then defines the mass and charge of the charging agent on the ions (e.g., proton, sodium ion, etc.) and a distribution which defines the average number of

charging agents per monomer unit. For example, if a polymer has on average a proton on every two to four monomer units, the distribution will range from 0.25 to 0.5. For consistency, this distribution also follows a gamma distribution and is defined with a mode and entropy. This approach will calculate fractional numbers of charge carriers for some polymer lengths, which is not physically possible; therefore, each fractional value is rounded both directions, which results in two *m/z* values corresponding to one polymer length with a particular number of charge carriers per monomer. Using the mass distributions, the polymer size distribution, and distribution of charge carriers per monomer unit, all possible mass and charge pairs can be calculated to generate a mass-to-charge spectrum.

7.4 References

[1] M. Mann, C.K. Meng, J.B. Fenn, Interpreting Mass Spectra of Multiply Charged Ions, Anal. Chem. 61 (1989) 1702–1708. https://doi.org/10.1021/ac00190a023.

[2] M.T. Marty, A.J. Baldwin, E.G. Marklund, G.K.A. Hochberg, J.L.P. Benesch, C. V. Robinson, Bayesian deconvolution of mass and ion mobility spectra: From binary interactions to polydisperse ensembles, Anal. Chem. 87 (2015) 4370–4376. https://doi.org/10.1021/acs.analchem.5b00140.

[3] S.P. Cleary, A.M. Thompson, J.S. Prell, Fourier Analysis Method for Analyzing Highly Congested Mass Spectra of Ion Populations with Repeated Subunits, Anal. Chem. 88 (2016) 6205–6213. https://doi.org/10.1021/acs.analchem.6b01088.

[4] S.A. McLuckey, J.L. Stephenson, Ion/ion chemistry of high-mass multiply charged ions, Mass Spectrom. Rev. 17 (1998) 369–407. https://doi.org/10.1002/(SICI)1098-2787(1998)17:6<369::AID-MAS1>3.0.CO;2-J.

[5] J.L. Stephenson, S.A. McLuckey, Simplification of Product Ion Spectra Derived from Multiply Charged Parent Ions via Ion/Ion Chemistry, Anal. Chem. 70 (1998) 3533–3544. https://doi.org/10.1021/ac9802832.

[6] K.J. Laszlo, M.F. Bush, Analysis of Native-Like Proteins and Protein Complexes Using Cation to Anion Proton Transfer Reactions (CAPTR), J. Am. Soc. Mass Spectrom. 26 (2015) 2152–2161. https://doi.org/10.1007/s13361-015-1245-4.

[7] M. He, S.A. McLuckey, Two ion/ion charge inversion steps to form a doubly protonated peptide from a singly protonated peptide in the gas phase, J. Am. Chem. Soc. 125 (2003) 7756–7757. https://doi.org/10.1021/ja0354521.

[8] H.P. Gunawardena, S.A. McLuckey, Synthesis of multi-unit protein hetero-complexes in the gas phase via ion–ion chemistry, J. Mass Spectrom. 39 (2004) 630–638. https://doi.org/10.1002/jms.629.

[9] M. Van De Waterbeemd, K.L. Fort, D. Boll, M. Reinhardt-Szyba, A. Routh, A. Makarov, A.J.R. Heck, High-fidelity mass analysis unveils heterogeneity in intact ribosomal particles, Nat. Methods. 14 (2017) 283–286. https://doi.org/10.1038/nmeth.4147.

[10] R Core Team, R: A Language and Environment for Statistical Computing, (2019). https://www.r-project.org/.

[11] W. Chang, J. Cheng, J. Allaire, Y. Xie, J. McPherson, shiny: Web Application Framework for R, (2019). https://cran.r-project.org/package=shiny.

[12] J. Owen, rhandsontable: Interface to the "Handsontable.js" Library, (2018). https://cran.r-project.org/package=rhandsontable.

[13] C. Sievert, Interactive Web-Based Data Visualization with R, plotly, and shiny, (2020). https://plotly-r.com.

[14] J. Ooms, The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects, (2014). http://arxiv.org/abs/1403.2805 (accessed May 27, 2020).

[15] H. Wickham, Reshaping data with the reshape package, J. Stat. Softw. 21 (2007) 1–20. https://doi.org/10.18637/jss.v021.i12.

[16] H. Wickham, R. François, L. Henry, K. Müller, dplyr: A Grammar of Data Manipulation, (2019). https://cran.r-project.org/package=dplyr.

APPENDIX A. SHIELDBUDDY AND ARDUINO DUE CODE AND SCHEMATICS FOR INSTRUMENT CONTROLLER

Shieldbuddy Code

```
#include<ArduinoJson.h>
#include<Wire.h>
#include<SoftwareWire.h>
SoftwareWire Wire3(SDA1, SCL1, true, false);

bool data_acquire = false;

/* Class declarations */
// Output parameter class
class Output
{
  private:
    uint8_t output_number;
    double start_frequency;
    double end_frequency;
    double current_frequency;
    double period_squared_step;
    const static double tuning_ratio = 4294967296.0 / 125000000;
    double duty_cycle;
    uint8_t tickle_div;
    double tickle_amplitude;
    uint8_t tickle_phase;
    const static uint8_t MCP1 = 40;
    const static uint8_t MCP2 = 41;
    const static uint8_t MCP3 = 42;
    const static uint8_t wiper0 = 0;
    const static uint8_t wiper1 = 16;
  public:
    Output(uint8_t output_index, JsonObject& parameters, uint32_t
num_frequency_steps);
    ~Output() {};
    void print();
    void updateDutyCycle();
    void updateTickle();
    void chooseSPIOutput();
    uint32_t nextFrequency();
    void resetFrequency();
    void updateFrequency(uint32_t frequency);
    uint32_t getFrequency();
};

// Class Initializer. Receives index value 0-2 to differentiate the three
outputs on the controller,
// list of parameters as a JsonObject, and the number of frequency steps for
a frequency ramp. Initializes
// all output variables and checks if they are appropriate numbers. If not
appropriate the parameter is set
// as 0.
```

```
Output::Output(uint8_t output_index, JsonObject& parameters, uint32_t
num_frequency_steps)
{
  output_number = output_index + 1;
  start_frequency = parameters["Start"];        // Frequency at beginning of
ramp
  end_frequency = parameters["End"];            // Frequency at end of ramp
  // The frequency is ramped according to the square of its inverse (the
period),
  // so the change in period-squared is calculated.
  period_squared_step = (1 / pow(end_frequency, 2) - 1 / pow(start_frequency,
2)) / num_frequency_steps;
  current_frequency = 1 / sqrt(1 / pow(start_frequency, 2) -
period_squared_step);  // Keeps track of frequency during ramp.

  // Parse duty cycle from parameters JsonObject and check that it's a valid
number.
  duty_cycle = parameters["Duty Cycle"];
  if(duty_cycle != duty_cycle)
  {
    duty_cycle = 0;
  }

  // Parse tickle amplitude from parameters and check that it's a valid
number.
  tickle_amplitude = parameters["Amplitude"];
  if(tickle_amplitude != tickle_amplitude)
  {
    tickle_amplitude = 0;
  }

  // Tickle phase is not implemented.
  tickle_phase = parameters["Phase"];
  if(tickle_phase != tickle_phase)
  {
    tickle_phase = 0;
  }

  // Parse tickle division from parameters. Can be divide by 2, 4, 8, 16.
  // Used for resonance ejection at different q values.
  // Or the tickle can come from Output #3 for a non-frequency locked tickle.
  // Used for CID at a specific q value.
  const char* tickle = parameters["Tickle"];
  if(strcmp(tickle, "Div / 2") == 0)
  {
    tickle_div = 2;
  }
  else if(strcmp(tickle, "Div / 4") == 0)
  {
    tickle_div = 4;
  }
  else if(strcmp(tickle, "Div / 8") == 0)
  {
    tickle_div = 8;
  }
  else if(strcmp(tickle, "Div / 16") == 0)
  {
```

```
    tickle_div = 16;
  }
  else if(strcmp(tickle, "Output 3") == 0)
  {
    tickle_div = 0;
  }
  else
  {
    tickle_div = 2;
  }
}

// Function that prints parameters to the computer screen when user asks to
// Upload the scan function.
void Output::print()
{
  SerialASC.println("--------------------------------------------------");
  SerialASC.print("Output:\t\t"); SerialASC.println(output_number);
  SerialASC.print("Start Frequency:\t"); SerialASC.println(start_frequency);
  SerialASC.print("End Frequency:\t\t"); SerialASC.println(end_frequency);
  SerialASC.print("Duty Cycle:\t\t"); SerialASC.println(duty_cycle);
  SerialASC.print("Tickle:\t\t"); SerialASC.println(tickle_div);
  SerialASC.print("Tickle Amplitude:\t");
SerialASC.println(tickle_amplitude);
  SerialASC.print("Tickle Phase:\t\t"); SerialASC.println(tickle_phase);
}

// Converts duty cycle parameter value 0-100 to a 8-bit number for the
// digital potentiometer that sets the duty cycle on the controller board.
uint8_t dutyCycleToAnalog(double duty_cycle)
{
  uint8_t analog_value = duty_cycle * 255 / 100;
  return analog_value;
}

// Tells the digital potentiometer to update the duty cycle
// over I2C using the Wire3 pins defined at the top of the code.
void Output::updateDutyCycle()
{
  // Each output has a different potentiometer. Addresses are stored
  // as MCP1, MCP2, MCP3.
  switch(output_number)
  {
    case 1:
      Wire3.beginTransmission(MCP1);
      break;
    case 2:
      Wire3.beginTransmission(MCP2);
      break;
    case 3:
      Wire3.beginTransmission(MCP3);
      break;
  }
  // wiper0 chooses the first potentiometer output which
  // controls duty cycle
  Wire3.write(wiper0);
  Wire3.write(dutyCycleToAnalog(duty_cycle));
```

```
  Wire3.endTransmission(false);
}

// Converts tickle amplitude 0-5 V to 8-bit number for
// the potentiometer.
uint8_t amplitudeToAnalog(double tickle_amplitude)
{
  uint8_t analog_value = (5 - tickle_amplitude) * 255 / 5;
  return analog_value;
}

// Updates tickle division and amplitude.
void Output::updateTickle()
{
  // Tickle division is controlled by digital outputs from the
  // ShieldBuddy. Each tickle output has three assigned digital
  // outputs. The first is LOW when using a division and HIGH when
  // using Output 3's tickle. The other two choose which value to
  // divide by with different combinations of LOW and HIGH.
  switch(output_number)
  {
    case 1:
      switch(tickle_div)
      {
        case 2:
          Fast_digitalWrite(33, LOW);
          Fast_digitalWrite(25, LOW);
          Fast_digitalWrite(26, LOW);
          break;
        case 4:
          Fast_digitalWrite(33, LOW);
          Fast_digitalWrite(25, HIGH);
          Fast_digitalWrite(26, LOW);
          break;
        case 8:
          Fast_digitalWrite(33, LOW);
          Fast_digitalWrite(25, LOW);
          Fast_digitalWrite(26, HIGH);
          break;
        case 16:
          Fast_digitalWrite(33, LOW);
          Fast_digitalWrite(25, HIGH);
          Fast_digitalWrite(26, HIGH);
          break;
        case 0:
          Fast_digitalWrite(33, HIGH);
          break;
      }
      Wire3.beginTransmission(MCP1);
      break;
    case 2:
      switch(tickle_div)
      {
        case 2:
          Fast_digitalWrite(34, LOW);
          Fast_digitalWrite(28, LOW);
          Fast_digitalWrite(29, LOW);
```

```
          break;
        case 4:
          Fast_digitalWrite(34, LOW);
          Fast_digitalWrite(28, HIGH);
          Fast_digitalWrite(29, LOW);
          break;
        case 8:
          Fast_digitalWrite(34, LOW);
          Fast_digitalWrite(28, LOW);
          Fast_digitalWrite(29, HIGH);
          break;
        case 16:
          Fast_digitalWrite(34, LOW);
          Fast_digitalWrite(28, HIGH);
          Fast_digitalWrite(29, HIGH);
          break;
        case 0:
          Fast_digitalWrite(34, HIGH);
          break;
      }
      Wire3.beginTransmission(MCP2);
      break;
    // Output 3 always has its own tickle, so there are only
    // two digital outputs to choose the division value.
    case 3:
      switch(tickle_div)
      {
        case 2:
          Fast_digitalWrite(31, LOW);
          Fast_digitalWrite(32, LOW);
          break;
        case 4:
          Fast_digitalWrite(31, HIGH);
          Fast_digitalWrite(32, LOW);
          break;
        case 8:
          Fast_digitalWrite(31, LOW);
          Fast_digitalWrite(32, HIGH);
          break;
        case 16:
          Fast_digitalWrite(31, HIGH);
          Fast_digitalWrite(32, HIGH);
          break;
      }
      Wire3.beginTransmission(MCP3);
      break;
  }
  // Addresses for the potentiometer are the same as with the duty cycle.
  // wiper1 chooses the second potentiometer output which controls tickle
  // amplitude.
  Wire3.write(wiper1);
  Wire3.write(amplitudeToAnalog(tickle_amplitude));
  Wire3.endTransmission(false);
}

// Two ShieldBuddy digital outputs control which waveform
// output will receive instructions for changing frequency over SPI.
```

```
void Output::chooseSPIOutput()
{
  switch(output_number)
  {
    case 1:
      Fast_digitalWrite(22, LOW);
      Fast_digitalWrite(23, LOW);
      break;
    case 2:
      Fast_digitalWrite(22, HIGH);
      Fast_digitalWrite(23, LOW);
      break;
    case 3:
      Fast_digitalWrite(22, LOW);
      Fast_digitalWrite(23, HIGH);
      break;
  }
}

// Resets the frequency to one step before the start frequency.
void Output::resetFrequency()
{
  current_frequency = 1 / sqrt(1 / pow(start_frequency, 2) -
period_squared_step);
}

// Calculates the next frequency using the period-squared step.
uint32_t Output::nextFrequency()
{
  current_frequency = 1 / sqrt(1 / pow(current_frequency, 2) +
period_squared_step);
  if(current_frequency != current_frequency)
  {
    current_frequency = 0;
  }
  uint32_t del_phase = current_frequency * tuning_ratio;
  return del_phase;
}

// Updates the frequency.
void Output::updateFrequency(uint32_t del_phase)
{
  // Establishes connection of SPI to chosen waveform output.
  chooseSPIOutput();
  // Sends 32-bit frequency information one bit at a time.
  // Digital output 63 tells the bit (0 or 1), and 62 triggers
  // a clock to acknowledge sending and receiving the bit defined
  // by 63.
  for(uint8_t i = 0; i < 32; i++)
  {
    Fast_digitalWrite(63, del_phase & 1)
    Fast_digitalWrite(62, HIGH);
    Fast_digitalWrite(62, LOW);
    del_phase >>= 1;
  }
  // The last 8 bits tell the change in phase for the square wave.
  // Probably should always be zero.
```

```
  for(uint8_t i = 0; i < 8; i++)
  {
    Fast_digitalWrite(63, 0);
    Fast_digitalWrite(62, HIGH);
    Fast_digitalWrite(62, LOW);
  }
}

// Used to request the current frequency for debugging purposes.
uint32_t Output::getFrequency()
{
  return (uint32_t)current_frequency;
}

// Scan function segment class
class Segment
{
  private:
    const static uint8_t num_outputs = 3;
    const static int DAC1 = 76;
    const static int DAC2 = 77;
    uint32_t duration;
    bool active;
    bool record;
    uint32_t num_freq_steps;
    const static uint8_t micros_per_step = 19;
    Output* output_list[num_outputs];
    uint8_t digital[12];
    double analog[8];
  public:
    Segment(JsonObject& segment);
    ~Segment();
    uint32_t getDuration();
    bool getActive();
    bool getRecord();
    uint32_t getNumSteps();
    void print();
    void setupSegment();
    void updateAnalogValue(uint8_t analog_output, double analog_volt);
    void updateAnalog();
    void updateOutputs();
    Output* getOutput(uint8_t output);
    void run();
    void stop();
};

// Class initializer. Receives parameters as JsonObject.
Segment::Segment(JsonObject& segment)
{
  // Parse duration of the segment in ms from parameters
  duration = segment["Duration"];
  // Parse whether segment is active or not from parameters.
  if(segment["Active"] == "True")
  {
    active = true;
  }
  else
```

```
  {
    active = false;
  }
  // Parse whether to record data during this segment.
  if(segment["Record"] == "True")
  {
    record = true;
  }
  else
  {
    record = false;
  }
  // Calculates the number of frequency steps for a ramp based on the
duration.
  num_freq_steps = duration * 1000 / micros_per_step;
  // Create the three Outputs for this segment.
  for(uint8_t i = 0; i < num_outputs; i++)
  {
    JsonObject& output_parameters = segment["Outputs"][i];
    output_list[i] = new Output(i, output_parameters, num_freq_steps);
  }
  // Parse instrument digital outputs from parameters.
  for(uint8_t i = 0; i < segment["Digital"].size(); i++)
  {
    if(segment["Digital"][i] == "True")
    {
      digital[i] = 1;
    }
    else
    {
      digital[i] = 0;
    }
  }
  // Parse instrument analog outputs from parameters.
  for(uint8_t i = 0; i < segment["Analog"].size(); i++)
  {
    analog[i] = segment["Analog"][i];
    if(analog[i] != analog[i])
    {
      analog[i] = 0;
    }
  }
}

// Destructor for segment class. Deletes outputs to free up memory
// when segment is deleted.
Segment::~Segment()
{
  for(uint8_t i = 0; i < num_outputs; i++)
  {
    delete output_list[i];
  }
}

// Used to check the segment duration.
uint32_t Segment::getDuration()
{
```

```
  return duration;
}

// Used to check if the segment is active or not.
bool Segment::getActive()
{
  return active;
}

// Used to check if data is recorded during this segment.
bool Segment::getRecord()
{
  return record;
}

// Used to get the number of frequency steps during this segment.
uint32_t Segment::getNumSteps()
{
  return num_freq_steps;
}

// Prints the segment variables to the computer over USB when user
// asks for a scan function upload.
void Segment::print()
{
  SerialASC.print("Duration:\t\t"); SerialASC.println(duration);
  SerialASC.print("Frequency steps:\t"); SerialASC.println(num_freq_steps);
  for(uint8_t i = 0; i < num_outputs; i++)
  {
    if(output_list[i] != NULL)
    {
      output_list[i]->print();
    }
    else
    {
      SerialASC.println("None");
    }
  }
}

// Prepares instrumer controller outputs for this segment prior to
// the frequency ramp.
void Segment::setupSegment()
{
  // Resets frequencies and updates them before starting ramps.
  for(uint8_t i = 0; i < num_outputs; i++)
  {
    output_list[i]->resetFrequency();
    output_list[i]->updateFrequency(output_list[i]->nextFrequency());
  }
  // Updates analog outputs for this segment.
  for(uint8_t i = 0; i < 8; i++)
  {
    updateAnalogValue(i, analog[i]);
  }
  // Updates duty cycles for this segment.
  for(uint8_t i = 0; i < num_outputs; i++)
```

```
  {
    output_list[i]->updateDutyCycle();
  }
  // Updates tickle parameters for this segment.
  for(uint8_t i = 0; i < num_outputs; i++)
  {
    output_list[i]->updateTickle();
  }
  // Updates digital outputs for this segment.
  // Digital outputs come directly from the ShieldBuddy.
  Fast_digitalWrite(42, digital[0]);
  Fast_digitalWrite(43, digital[1]);
  Fast_digitalWrite(44, digital[2]);
  Fast_digitalWrite(45, digital[3]);
  Fast_digitalWrite(46, digital[4]);
  Fast_digitalWrite(47, digital[5]);
  Fast_digitalWrite(48, digital[6]);
  Fast_digitalWrite(49, digital[7]);
  Fast_digitalWrite(50A, digital[8]);
  Fast_digitalWrite(51A, digital[9]);
  Fast_digitalWrite(52A, digital[10]);
  Fast_digitalWrite(53A, digital[11]);
  // Triggers all analog outputs to update simultaneously,
  // closely followed by all waveform frequencies being updated
  // nearly simultaneously.
  updateAnalog();
  updateOutputs();
}

// Analog outputs are controlled by digital to analog converters (DAC).
// This function stores values in DAC memory but does not change the DAC
// outputs.
void Segment::updateAnalogValue(uint8_t analog_output, double analog_volt)
{
  // First four analog outputs 0-3 are controlled by DAC1
  if (analog_output < 4)
  {
    Wire.beginTransmission(DAC1);           // Call address stored as DAC1
    // DAC has four channels identified as 0, 2, 4, 6
    Wire.write((analog_output % 4) * 2);
    Wire.write(highByte(voltToAnalog(analog_volt)));
    Wire.write(lowByte(voltToAnalog(analog_volt)));
    Wire.endTransmission(false);
  }
  // Last four analog outputs 4-7 are controlled by DAC2.
  else if (analog_output >= 4)
  {
    Wire.beginTransmission(DAC2);
    Wire.write((analog_output % 4) * 2);
    Wire.write(highByte(voltToAnalog(analog_volt)));
    Wire.write(lowByte(voltToAnalog(analog_volt)));
    Wire.endTransmission(false);
  }
}

// Tells all DACs to change analog outputs to values stored
// in memory.
```

```
void Segment::updateAnalog()
{
  Wire.beginTransmission(72);   // General address for DAC1 and DAC2.
  Wire.write(48);               // Tells all four DAC outputs to change.
  Wire.write(1);
  Wire.write(1);
  Wire.endTransmission(false);
}

// Calculates 16-bit instruction for DAC from desired
// analog output value.
uint16_t voltToAnalog(double volt)
{
  uint16_t analog_value = volt * 3281 + 33379;
  return analog_value;
}

// Updates all waveform outputs nearly simultaneously.
void Segment::updateOutputs()
{
  Fast_digitalWrite(9, HIGH);
  Fast_digitalWrite(7, HIGH);
  Fast_digitalWrite(5, HIGH);
  Fast_digitalWrite(9, LOW);
  Fast_digitalWrite(7, LOW);
  Fast_digitalWrite(5, LOW);
}

// Used to check parameters of the waveform outputs.
Output* Segment::getOutput(uint8_t output)
{
  return output_list[output];
}

// Updates frequencies of all three waveform outputs.
// This function is called repeatedly during a frequency ramp.
void Segment::run()
{
  for(uint8_t i = 0; i < num_outputs; i++)
  {
    output_list[i]->updateFrequency(output_list[i]->nextFrequency());
  }
  updateOutputs();
}

// Sets all digital and analog outputs and frequencies to zero
// when user tells scan function to stop.
void Segment::stop()
{
  for(uint8_t i = 0; i < 12; i++)
  {
    digitalWrite(i + 42, LOW);
  }
  for(uint8_t i = 0; i < 8; i++)
  {
    updateAnalogValue(i, 0);
  }
```

```
  updateAnalog();
  for(uint8_t i = 0; i < num_outputs; i++)
  {
    output_list[i]->updateFrequency(0);
  }
  updateOutputs();
}

// Scan function class - container for the scan function segment objects
class ScanFunction
{
  private:
    uint8_t current_size;
    const static uint8_t max_size = 20;
    Segment* segment_list[max_size];
  public:
    ScanFunction();
    ~ScanFunction() {};
    void addSegment(JsonObject& segment);
    uint32_t getSegmentDuration(uint8_t segment_index);
    bool getSegmentRecord(uint8_t segment_index);
    uint8_t size();
    void print();
    void clear();
    void run();
    void stop();
};

// Scan function initialized with a size of 0.
ScanFunction::ScanFunction()
{
  current_size = 0;
}

// Creates a new segment from the parameters JsonObject.
void ScanFunction::addSegment(JsonObject& segment)
{
  // Maximum size of scan function is 20 to protect memory.
  if(current_size < max_size)
  {
    Segment* new_segment = new Segment(segment);
    // Checks if segment is defined as active.
    // If not, it gets deleted from memory.
    if(new_segment->getActive())
    {
      segment_list[current_size] = new_segment;
      current_size++;
    }
    else
    {
      delete new_segment;
    }
  }
  else
  {
    SerialASC.print("Scan function length exceeds maximum size of ");
SerialASC.println(max_size);
```

```
  }
}

// Used to check the duration of a chosen segment.
uint32_t ScanFunction::getSegmentDuration(uint8_t segment_index)
{
  return segment_list[segment_index]->getDuration();
}

// Used to check if data is recorded during a chosen segment.
bool ScanFunction::getSegmentRecord(uint8_t segment_index)
{
  return segment_list[segment_index]->getRecord();
}

// Used to check the size of the scan function.
uint8_t ScanFunction::size()
{
  return current_size;
}

// Prints the scan function parameters to computer when user asks to upload
scan function.
void ScanFunction::print()
{
  for(uint8_t i = 0; i < current_size; i++)
  {
    SerialASC.println("-----------------------------------------------------");
    SerialASC.print("Segment:\t\t"); SerialASC.println(i + 1);
    segment_list[i]->print();
    SerialASC.println("-----------------------------------------------------");
  }
}

// Deletes all segments to clear memory for a new scan function.
// Resets size to 0.
void ScanFunction::clear()
{
  for(uint8_t i = 0; i < current_size; i++)
  {
    delete segment_list[i];
  }
  current_size = 0;
}

// Runs scan function segments in order.
void ScanFunction::run()
{
  for(uint8_t i = 0; i < current_size; i++)
  {
    // Updates outputs for new segment before starting frequency ramp.
    segment_list[i]->setupSegment();
    // Check number of frequency steps for ramp.
    uint32_t num_steps = segment_list[i]->getNumSteps();
    // If recording during this segment trigger Arduino Due to start
    // sending data to computer with ShieldBuddy digital pin 13.
    if(segment_list[i]->getRecord())
```

```
    {
      Fast_digitalWrite(13, 1);
    }
    // Iterate through the number of frequency steps.
    // Update frequencies at each iteration.
    for(uint32_t j = 0; j < num_steps; j++)
    {
      segment_list[i]->run();
    }
    // If recording during this segment trigger Arduino Due to stop
    // sending data to computer.
    if(segment_list[i]->getRecord())
    {
      Fast_digitalWrite(13, 0);
      // Prints final frequency of waveform output 2 to computer.
      // Useful for checking if frequency ramp ended at correct end
frequency.
      SerialASC.print(segment_list[i]->getOutput(1)->getFrequency());
    }
  }
}

// Stops scan function.
void ScanFunction::stop()
{
  if(current_size > 0)
  {
    segment_list[0]->stop();
  }
}

/*** Don't worry, the normal Arduino setup() and loop() are below this block!
***/

/* LMU uninitialised data */
StartOfUninitialised_LMURam_Variables
/* Put your LMU RAM fast access variables that have no initial values here
e.g. uint32 LMU_var; */
EndOfUninitialised_LMURam_Variables

/* LMU uninitialised data */
StartOfInitialised_LMURam_Variables
/* Put your LMU RAM fast access variables that have an initial value here
e.g. uint32 LMU_var_init = 1; */
EndOfInitialised_LMURam_Variables

/* If you do not care where variables end up, declare them here! */
const uint32_t SERIAL_RATE = 2000000;
const uint16_t SERIAL_TIMEOUT = 5000;
const uint16_t MAX_INPUT_LENGTH = 10000;
ScanFunction scan_function;


/*** Core 0 ***/

void setup() {
  // put your setup code for core 0 here, to run once:
```

```
  SerialASC.begin(SERIAL_RATE);              // Start USB connection with
computer.
  SerialASC.setTimeout(SERIAL_TIMEOUT);      // USB communication will give up
after 5 s.
  // ShieldBuddy pins 62 and 63 are for SPI communication with waveform
outputs.
  pinMode(62, OUTPUT);
  pinMode(63, OUTPUT);
  // Wire3 is I2C communication with potentiometers for duty cycle
  // and tickle amplitude.
  Wire3.begin();
  Wire3.setClock(152435);
  // Wire is I2C communication with DACs for controller analog outputs.
  Wire.setWireBaudrate(400000);
  Wire.begin();
  // ShieldBuddy pins used for controller digital outputs.
  for(int i = 22; i < 54; i++)
  {
    pinMode(i, OUTPUT);
    digitalWrite(i, LOW);
  }
  // ShieldBuddy pins used for establishing connections with the different
  // waveform outputs.
  for(int i = 4; i < 12; i++)
  {
    pinMode(i, OUTPUT);
  }
  Fast_digitalWrite(10, HIGH);  // Disconnected, needs to be HIGH for some
reason.
  // The following prepares waveform outputs for use.
  // They are each reset, told to acknowledge receipt of a new frequency
(even though
  // none was sent), and then updated.
  Fast_digitalWrite(8, HIGH);   // Resets waveform output 1
  Fast_digitalWrite(8, LOW);
  Fast_digitalWrite(22, LOW);   // Establish connection with waveform output
1.
  Fast_digitalWrite(23, LOW);
  Fast_digitalWrite(62, HIGH);  // Trigger frequency data acknowledge.
  Fast_digitalWrite(62, LOW);
  Fast_digitalWrite(9, HIGH);   // Update waveform output 1.
  Fast_digitalWrite(9, LOW);
  Fast_digitalWrite(6, HIGH);   // Resets waveform output 2.
  Fast_digitalWrite(6, LOW);
  Fast_digitalWrite(22, HIGH);  // Establish connection with waveform output
2.
  Fast_digitalWrite(23, LOW);
  Fast_digitalWrite(62, HIGH);  // Data acknowledge
  Fast_digitalWrite(62, LOW);
  Fast_digitalWrite(7, HIGH);   // Update waveform output 2.
  Fast_digitalWrite(7, LOW);
  Fast_digitalWrite(4, HIGH);   // Resets waveform output 3.
  Fast_digitalWrite(4, LOW);
  Fast_digitalWrite(22, LOW);   // Establish connection with waveform output
3.
  Fast_digitalWrite(23, HIGH);
  Fast_digitalWrite(62, HIGH);  // Data acknowledge
```

```
  Fast_digitalWrite(62, LOW);
  Fast_digitalWrite(5, HIGH);   // Update waveform output 3.
  Fast_digitalWrite(5, LOW);

  SerialASC.println("Setup complete");
}


void loop()
{
  // Check is computer is saying something.
  if(SerialASC.available())
  {
    // Computer will send a letter depending on which button was clicked.
    char choice = SerialASC.read();
    switch(choice)
    {
      case 'D':
        downloadScan();
        break;
      case 'U':
        uploadScan();
        break;
      case 'R':
        runScan();
        break;
      case 'S':
        stopScan();
        break;
    }
  }
}


/*** Core 1 ***/

/* CPU1 Uninitialised Data */
StartOfUninitialised_CPU1_Variables
/* Put your CPU1 fast access variables that have no initial values here e.g.
uint32 CPU1_var; */
//uint32_t record_duration;
EndOfUninitialised_CPU1_Variables

/* CPU1 Initialised Data */
StartOfInitialised_CPU1_Variables
/* Put your CPU1 fast access variables that have an initial value here e.g.
uint32 CPU1_var_init = 1; */
EndOfInitialised_CPU1_Variables

void setup1() {
  // put your setup code for core 1 here, to run once:

  // ShieldBuddy pin 13 used to trigger Arduino Due for data collection.
  // This could be in the normal setup.
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
}
```

```
void loop1() {


}



/*** Core 2 ***/

/* CPU2 Uninitialised Data */
StartOfUninitialised_CPU2_Variables
/* Put your CPU2 fast access variables that have no initial values here e.g.
uint32 CPU2_var; */
EndOfUninitialised_CPU2_Variables

/* CPU2 Initialised Data */
StartOfInitialised_CPU2_Variables
/* Put your CPU2 fast access variables that have an initial value here e.g.
uint32 CPU2_var_init = 1; */
EndOfInitialised_CPU2_Variables


void setup2() {
  // put your setup code for core 2 here, to run once:


}


void loop2() {
  // put your main code for core 2 here, to run repeatedly:


}



/* Controller functions */
// Downloads scan parameters from computer.
void downloadScan()
{
  SerialASC.println("Download initiated");
  scan_function.clear();
  SerialASC.read();          // Read opening '[' from json array before
parsing objects
  char next_char = ',';      // Comma separates json objects in array
  while(next_char == ',')
  {
    // Create buffer for holding parameters until scan function is made.
    StaticJsonBuffer<MAX_INPUT_LENGTH> json_buffer;
    // Parse a segment.
    JsonObject& scan_segment = json_buffer.parseObject(SerialASC);
    if(scan_segment.success())
    {
      scan_function.addSegment(scan_segment);
```

```
      if(SerialASC.available())
      {
        next_char = SerialASC.read(); // Check if another comma - signals
there is another segment to read
      }
    }
    else
    {
      SerialASC.println("Download failed");
      next_char = '0';  // End while loop if bad segment
    }
    if(next_char == ']'){SerialASC.println("Download successful");}
  }
  while(SerialASC.available()){SerialASC.read();} // Clear input buffer
  SerialASC.println(scan_function.size());
  SerialASC.println("Download finished");
  return;
}

// Sends scan function back to computer. Checks if scan function was
downloaded correctly.
void uploadScan()
{
  SerialASC.println("Upload initiated");
  scan_function.print();
  SerialASC.println("Upload finished");
  return;
}

// Start scan function.
void runScan()
{
  SerialASC.println("Running scan function");
  char choice = ' ';
  // Keep checking for a stop signal from computer.
  while(choice != 'S')
  {
    scan_function.run();
    if(SerialASC.available())
    {
      choice = SerialASC.read();
    }
  }
  stopScan();
  return;
}

// Stops scan function.
void stopScan()
{
  SerialASC.println("Stopping scan function");
  scan_function.stop();
  return;
}
```

Arduino Due Code

The following code was lightly modified from code written by an online user named Stimmer. The code tells an Arduino Due to act as a 1 mega sample per second (MSPS) digitizer for fast data acquisition. Modifications were included to include triggering data transfer through USB to a computer, allowing mass spectra to be recorded.

```
#undef HID_ENABLED

// Arduino Due ADC->DMA->USB 1MSPS
// by stimmer
// from http://forum.arduino.cc/index.php?topic=137635.msg1136315#msg1136315
// Input: Analog in A0
// Output: Raw stream of uint16_t in range 0-4095 on Native USB Serial/ACM

// on linux, to stop the OS cooking your data:
// stty -F /dev/ttyACM0 raw -iexten -echo -echoe -echok -echoctl -echoke -
onlcr

volatile int bufn,obufn;
uint16_t buf[4][256];   // 4 buffers of 256 readings

void ADC_Handler(){     // move DMA pointers to next buffer
  int f=ADC->ADC_ISR;
  if (f&(1<<27)){
    bufn=(bufn+1)%4;
    ADC->ADC_RNPR=(uint32_t)buf[bufn];
    ADC->ADC_RNCR=256;
  }
}

void setup(){
  SerialUSB.begin(0);
  while(!SerialUSB);
  pmc_enable_periph_clk(ID_ADC);
  adc_init(ADC, SystemCoreClock, 21000000L, ADC_STARTUP_FAST);
//  ADC->ADC_MR |=0x80; // free running
//  ADC->ADC_MR |= ADC_TRIG_EXT;

  ADC->ADC_CHER=0x80;

  NVIC_EnableIRQ(ADC_IRQn);
  ADC->ADC_IDR=~(1<<27);
  ADC->ADC_IER=1<<27;
  ADC->ADC_RPR=(uint32_t)buf[0];   // DMA buffer
  ADC->ADC_RCR=256;
  ADC->ADC_RNPR=(uint32_t)buf[1]; // next DMA buffer
  ADC->ADC_RNCR=256;
  obufn=0;
  bufn=1;
  ADC->ADC_PTCR=1;
  ADC->ADC_CR=2;

  pinMode(13, INPUT);
```

```
}
const byte end_byte[4] = {'s', 't', 'o', 'p'};
void loop(){
//  ADC->ADC_MR=0x00000000;
  while((PIOB->PIO_PDSR & 1<<27) != 1<<27);
  ADC->ADC_MR |=0x80; // free running
  while((PIOB->PIO_PDSR & 1<<27) == 1<<27) {
//    while(obufn==bufn); // wait for buffer to be full
    while((obufn+1)%4==bufn); // wait for buffer to be full
    SerialUSB.write((uint8_t *)buf[obufn],512); // send it - 512 bytes = 256
uint16_t
    obufn=(obufn+1)%4;
  }
  SerialUSB.write(end_byte, 4);
  ADC->ADC_MR &=0xFFFFFF7F;
}
```

Instrument Controller Schematics

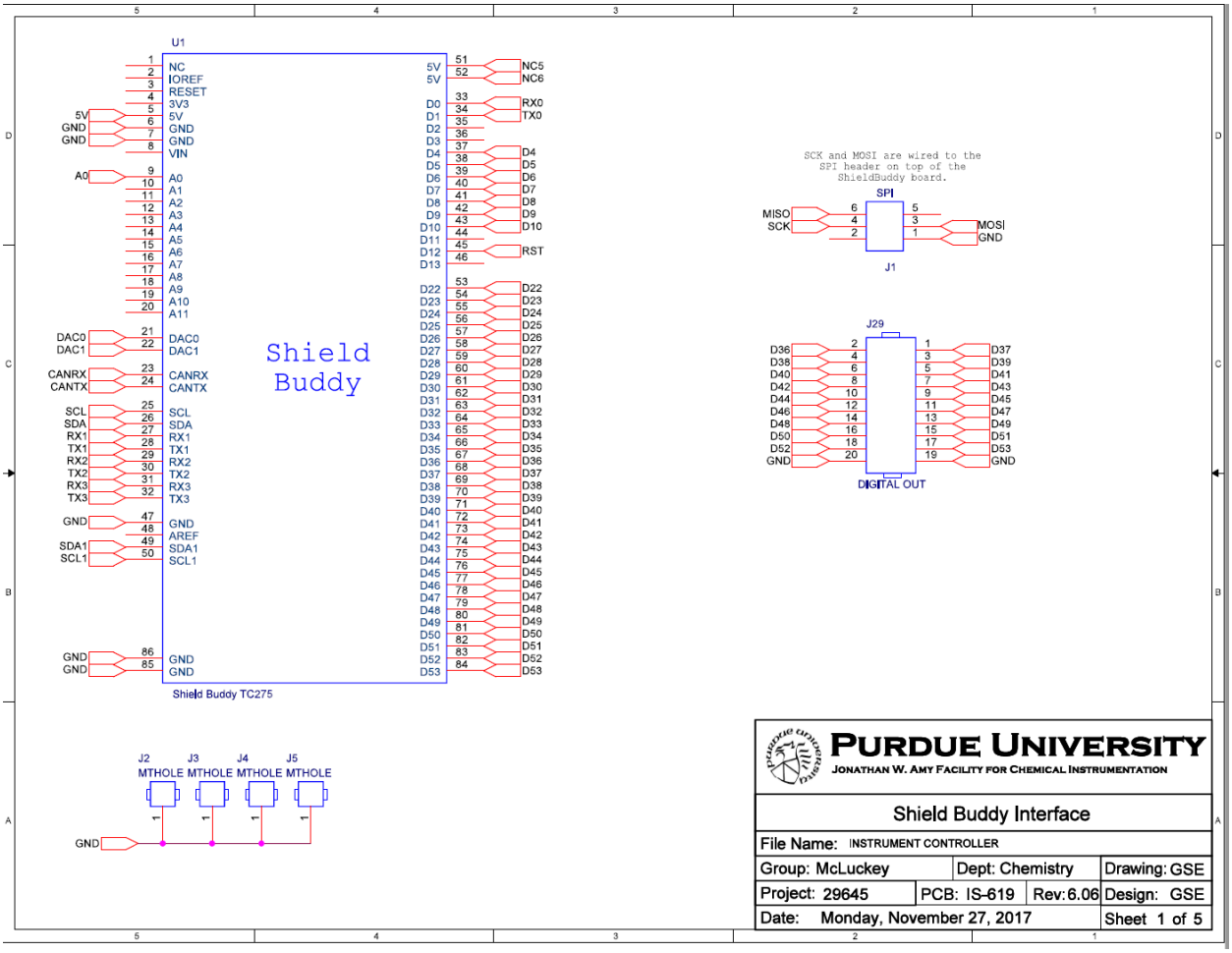

Figure A.1. Schematic of ShieldBuddy microcontroller pins (U1). The SPI (Serial Peripheral Interface) connector (J1) is shown separated from the ShieldBuddy. Digital outputs D36–D53 are routed to a connector (J29) to be connected to BNC connectors.

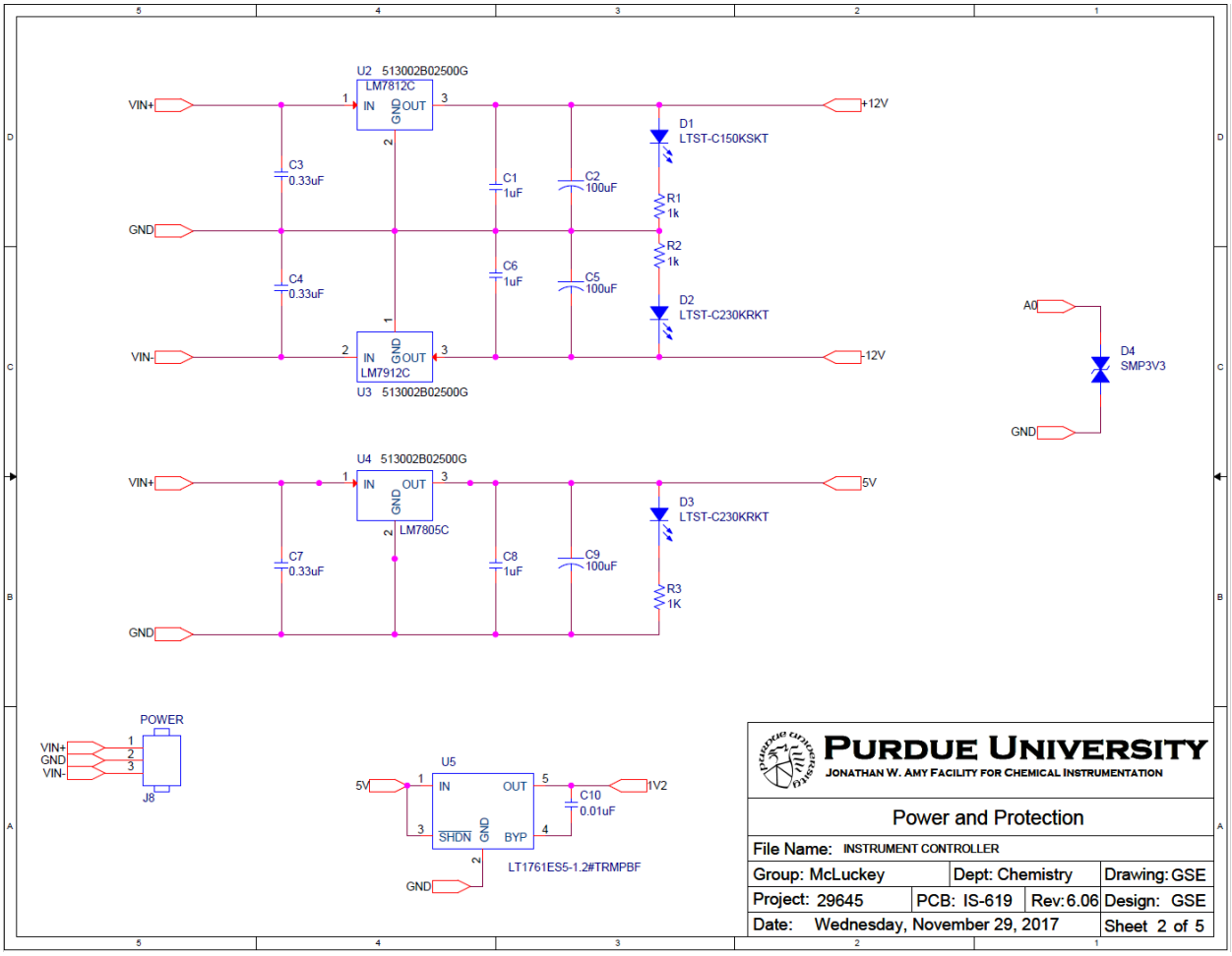

Figure A.2. Schematic of power input to main PCB (J8). VIN+ should be 15V and VIN− should be −15V.

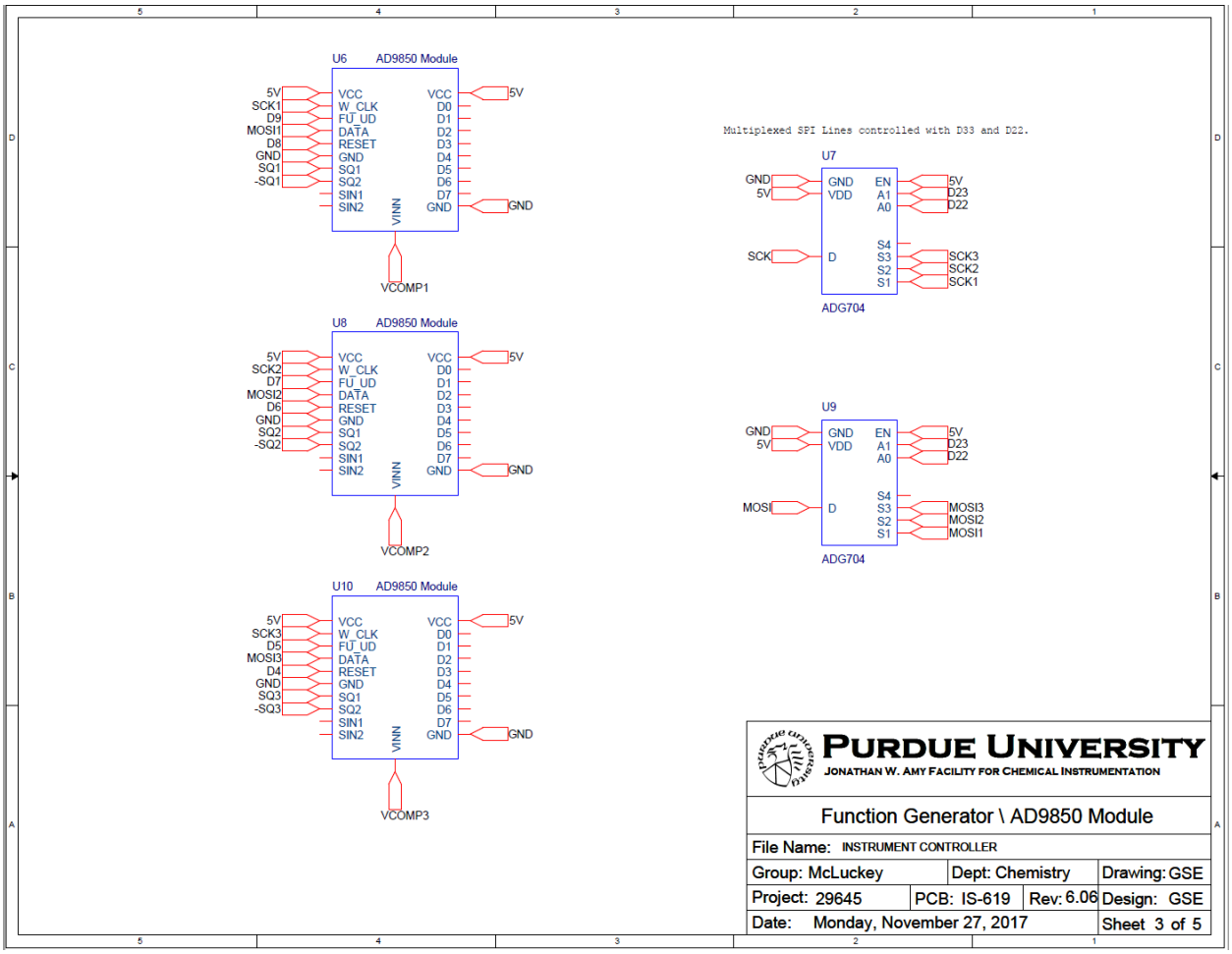

Figure A.3. Schematic of waveform generator modules (U6, U8, U10). Communication from the ShieldBuddy is managed by two multiplexers (U7 and U9) which switches the clock signal and data lines to the specified AD9850 module.

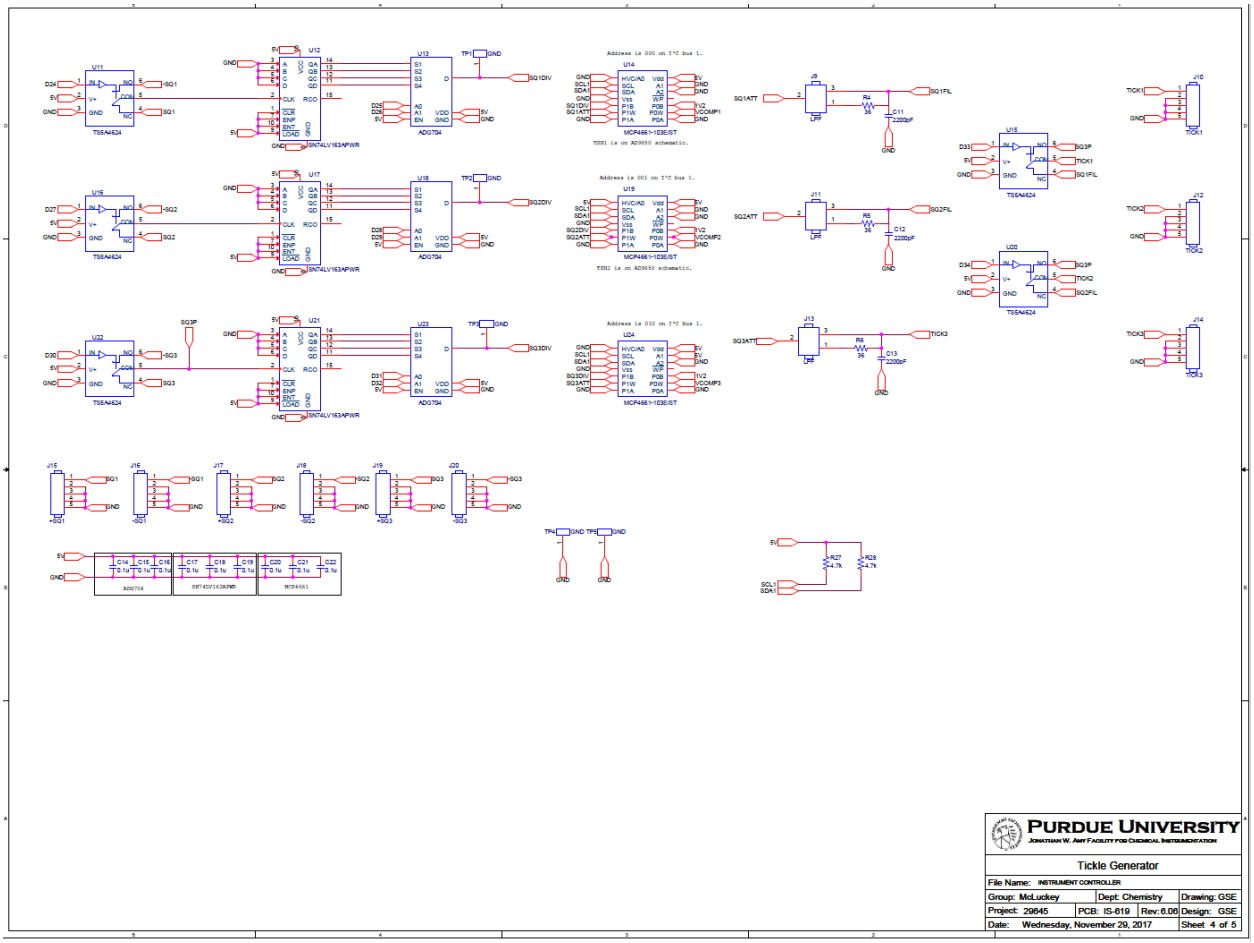

Figure A.4. Schematic for generating the frequency divided dipolar excitation waveforms. Each of the three outputs initially come from one of the three AD9850 modules in **Error! Reference source not found.**. The divided excitation waveform can be a frequency division of 2, 4, 8, or 16. Additionally, the third outputs can be routed to be the excitation for either the first or second AD9850 module. The non-frequency divided excitation is useful for excitation at an arbitrary frequency (e.g., exciting a specific trapped ion for CID).

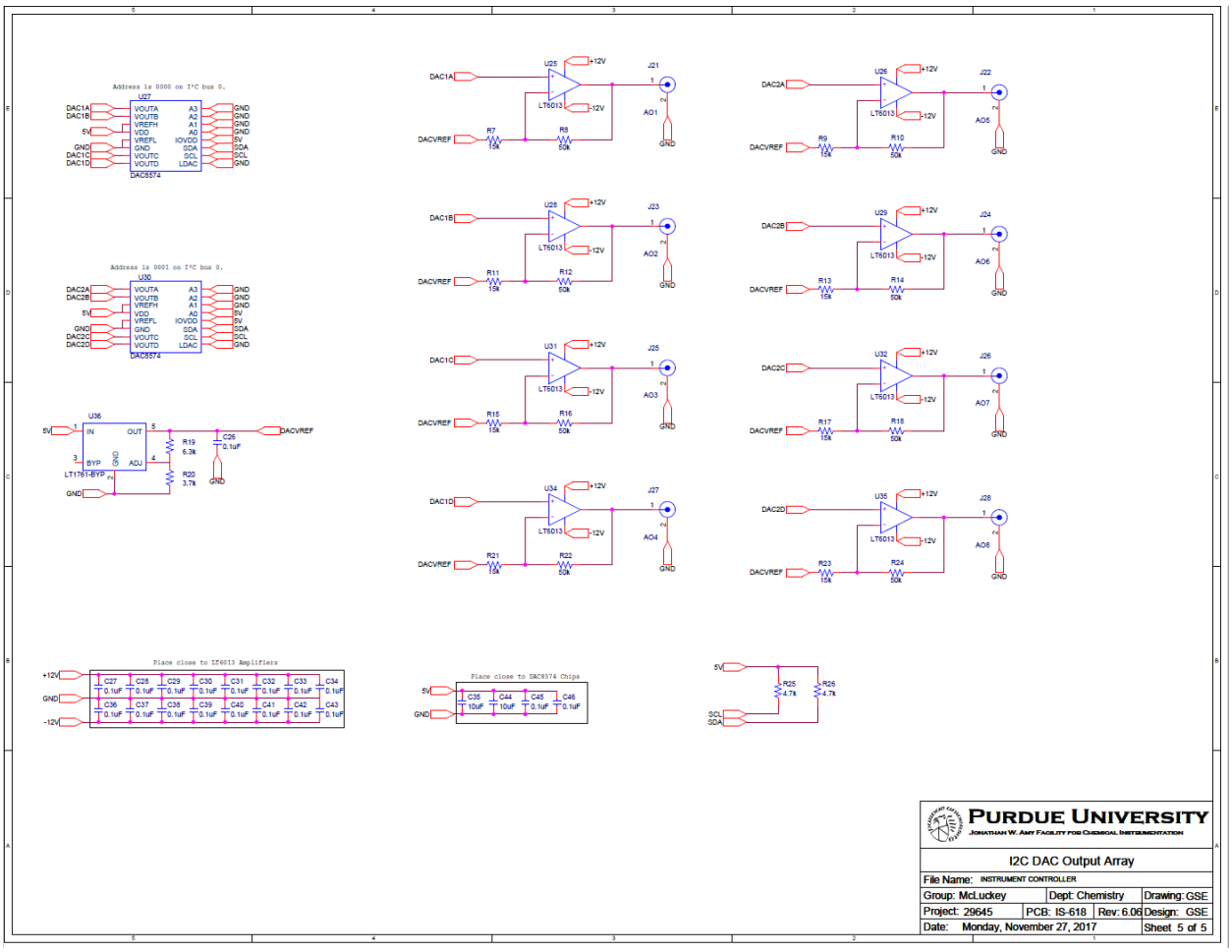

Figure A.5. Schematic for analog outputs controlled by two DACs (digital to analog converters). Each is designed to have a −10 to 10 V range.

APPENDIX B. PYTHON CODE FOR INSTRUMENT CONTROLLER SOFTWARE

"main.py"

```
import sys
from PyQt5.QtWidgets import QApplication
from MainWindow import MainWindow

def main():
    app = QApplication(sys.argv)
    mainWindow = MainWindow()
    mainWindow.move(app.screens()[0].geometry().topLeft())
    mainWindow.showMaximized()
    mainWindow.dataWindow.move(app.screens()[-1].geometry().topLeft())
    mainWindow.dataWindow.showMaximized()
    sys.exit(app.exec())

if __name__ == '__main__':
    main()
```

"MainWindow.py"

```
from PyQt5.QtWidgets import QMainWindow, QWidget, QGridLayout, QMessageBox,
QPushButton, QPlainTextEdit
from DialogWindows import ConnectionWindow, DataSettingsWindow,
AddRemoveSegmentWindow, CalculatorWindow
from DataWindow import DataWindow
from ScanFunction import ScanWidget
from serial import SerialException
from time import sleep
import json


class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()

        self.setWindowTitle("Scan Function Creator")

        self.textWidget = TextWidget()
        self.scanWidget = ScanWidget(self.textWidget)
        self.btnWidget = BtnWidget()

        self.connectWindow = ConnectionWindow(self.textWidget)
        self.dataSettingsWindow = DataSettingsWindow()
        self.addRemoveWindow = AddRemoveSegmentWindow()
        self.calcWindow = CalculatorWindow()

        self.dataWindow = DataWindow()

        self.build_menu()
```

```
        self.build_window()

        self.signal_handler()

    def build_menu(self):
        self.menuBar()

        self.fileMenu = self.menuBar().addMenu("File")
        self.openAction = self.fileMenu.addAction("Open Scan")
        self.saveAction = self.fileMenu.addAction("Save Scan")

        self.editMenu = self.menuBar().addMenu("Edit")
        self.addRemoveAction = self.editMenu.addAction("Add/Remove segments")
        self.calcAction = self.editMenu.addAction("Calculator")
        self.calibrateAction = self.editMenu.addAction("Calibrate plot")

        self.settingsMenu = self.menuBar().addMenu("Settings")
        self.connectAction = self.settingsMenu.addAction("Connect")
        self.dataSettingsAction = self.settingsMenu.addAction("Data
Settings")

    def build_window(self):
        self.setCentralWidget(QWidget())
        self.centralWidget().setLayout(QGridLayout())

        self.centralWidget().layout().setColumnStretch(0, 1)
        self.centralWidget().layout().setColumnStretch(1, 0)

        self.centralWidget().layout().addWidget(self.scanWidget, 0, 0, 2, 1)
        self.centralWidget().layout().addWidget(self.btnWidget, 0, 1)
        self.centralWidget().layout().addWidget(self.textWidget, 1, 1)

    def signal_handler(self):
        self.saveAction.triggered.connect(self.scanWidget.save_scan)
        self.openAction.triggered.connect(self.scanWidget.open_scan)

        self.addRemoveAction.triggered.connect(self.addRemoveWindow.show)
        self.calcAction.triggered.connect(self.calcWindow.show)
        self.connectAction.triggered.connect(self.connectWindow.show)

self.dataSettingsAction.triggered.connect(self.dataSettingsWindow.show)

        self.addRemoveWindow.addSegBtn.clicked.connect(lambda:
self.scanWidget.scanArea.add_segment(int(self.addRemoveWindow.addPositionBox.
text()) - 1))
        self.addRemoveWindow.removeSegBtn.clicked.connect(lambda:
self.scanWidget.scanArea.remove_segment(int(self.addRemoveWindow.removePositi
onBox.text()) - 1))


self.connectWindow.dataThread.dataSignal.connect(self.dataWindow.dataPlot.upd
ate)

self.connectWindow.dataThread.textSignal.connect(self.textWidget.appendPlainT
ext)
```

```
        self.dataSettingsWindow.applyBtn.clicked.connect(lambda:
self.dataWindow.dataPlot.set_sample(self.dataSettingsWindow.dataSampleBox.tex
t()))


self.calcWindow.updated.connect(self.dataWindow.dataToolWidget.constBox.setTe
xt)

self.calcWindow.updated.connect(self.dataWindow.displayToolWidget.constBox.se
tText)
        self.calcWindow.updated.emit(str(self.calcWindow.constant))

        self.btnWidget.runBtn.clicked.connect(self.run_scan)
        self.btnWidget.stopBtn.clicked.connect(self.stop_scan)
        self.btnWidget.downloadBtn.clicked.connect(self.download_scan)
        self.btnWidget.uploadBtn.clicked.connect(self.upload_scan)

    def closeEvent(self, event):
        event.ignore()
        choice = self.scanWidget.save_check()
        if choice == QMessageBox.Cancel:
            pass
        else:
            if choice == QMessageBox.Yes:
                self.scanWidget.save_scan()
            self.dataWindow.isClosable = True
            self.dataWindow.close()
            event.accept()

    def download_scan(self):
        try:
            scanData = json.dumps(self.scanWidget.scanFunction)
            self.connectWindow.controlPort.serial_write('D')
            self.connectWindow.controlPort.serial_write(scanData)
        except SerialException:
            self.textWidget.appendPlainText("No serial port found")

    def upload_scan(self):
        try:
            self.connectWindow.controlPort.serial_write('U')
        except SerialException:
            self.textWidget.appendPlainText("No serial port found")

    def run_scan(self):
        try:
            self.connectWindow.controlPort.serial_write('R')
            self.btnWidget.downloadBtn.setEnabled(False)
            self.btnWidget.uploadBtn.setEnabled(False)
            self.btnWidget.runBtn.setEnabled(False)
            self.btnWidget.stopBtn.setEnabled(True)
        except SerialException:
            self.textWidget.appendPlainText("No serial port found")

    def stop_scan(self):
        try:
            sleep(1)
            self.connectWindow.controlPort.serial_write('S')
```

```
            self.connectWindow.controlPort.reset_input_buffer()
            self.btnWidget.downloadBtn.setEnabled(True)
            self.btnWidget.uploadBtn.setEnabled(True)
            self.btnWidget.runBtn.setEnabled(True)
            self.btnWidget.stopBtn.setEnabled(False)
        except SerialException:
            self.textWidget.appendPlainText("No serial port found")

class BtnWidget(QWidget):
    def __init__(self):
        super(BtnWidget, self).__init__()

        self.downloadBtn = QPushButton("Download Scan")
        self.uploadBtn = QPushButton("Upload Scan")
        self.runBtn = QPushButton("Run Scan")
        self.stopBtn = QPushButton("Stop Scan")

        self.build_widget()

    def build_widget(self):
        self.setLayout(QGridLayout())

        self.layout().addWidget(self.downloadBtn, 0, 0)
        self.layout().addWidget(self.uploadBtn, 0, 1)
        self.layout().addWidget(self.runBtn, 1, 0)
        self.layout().addWidget(self.stopBtn, 1, 1)
        self.stopBtn.setEnabled(False)

class TextWidget(QPlainTextEdit):
    def __init__(self):
        super(TextWidget, self).__init__()

        self.setReadOnly(True)
```

“DataWindow.py”

```
from PyQt5.QtWidgets import QMainWindow, QWidget, QGridLayout, QPushButton,
QLabel, QLineEdit, QFileDialog
from PyQt5.QtCore import pyqtSignal
import pyqtgraph as pg

pg.setConfigOption('background', 'w')
pg.setConfigOption('foreground', 'k')

class DataWindow(QMainWindow):
    def __init__(self):
        super(DataWindow, self).__init__()

        self.setWindowTitle("Data Collector and Viewer")

        self.isClosable = False

        self.dataToolWidget = DataToolWidget()
        self.dataPlot = DataPlot()
```

```
        self.displayToolWidget = DisplayToolWidget()
        self.displayPlot = DisplayPlot()
        self.integralToolWidget = IntegralToolWidget()
        self.integralPlot = IntegralPlot()

        self.build_window()

        self.signal_handler()

        self.show()
        self.setGeometry(0,0,1000,800)

    def build_window(self):
        self.setCentralWidget(QWidget())
        self.centralWidget().setLayout(QGridLayout())

        self.centralWidget().layout().addWidget(self.dataToolWidget, 0, 0)
        self.centralWidget().layout().addWidget(self.dataPlot, 1, 0)
        self.centralWidget().layout().addWidget(self.displayToolWidget, 2, 0)
        self.centralWidget().layout().addWidget(self.displayPlot, 3, 0)
        self.centralWidget().layout().addWidget(self.integralToolWidget, 4,
0)
        self.centralWidget().layout().addWidget(self.integralPlot, 5, 0)

    def signal_handler(self):
        self.dataToolWidget.saveBtn.clicked.connect(self.dataPlot.save_data)

self.dataToolWidget.averagesBox.textChanged.connect(self.dataPlot.set_average
s)

self.dataPlot.countSignal.connect(self.dataToolWidget.countBox.setText)
        self.dataPlot.integralSignal.connect(self.integralPlot.update)
        self.dataToolWidget.calibrateBtn.clicked.connect(lambda:
self.dataPlot.calibrate(self.dataToolWidget.constBox.text(),
self.dataToolWidget.startFreqBox.text(),
self.dataToolWidget.endFreqBox.text()))


self.displayToolWidget.openBtn.clicked.connect(self.displayPlot.open_data)

self.displayToolWidget.saveBtn.clicked.connect(self.displayPlot.save_data)
        self.displayToolWidget.calibrateBtn.clicked.connect(lambda:
self.displayPlot.calibrate(self.displayToolWidget.constBox.text(),
self.displayToolWidget.startFreqBox.text(),
self.displayToolWidget.endFreqBox.text()))


self.integralToolWidget.clearBtn.clicked.connect(self.integralPlot.clr)

    def closeEvent(self, event):
        if self.isClosable:
            event.accept()
        else:
            event.ignore()

class DataToolWidget(QWidget):
    def __init__(self):
```

```
        super(DataToolWidget, self).__init__()

        self.build_widget()

    def build_widget(self):
        self.setLayout(QGridLayout())

        self.saveBtn = QPushButton("Save")
        self.layout().addWidget(self.saveBtn, 0, 0)

        self.layout().addWidget(QLabel("Averages"), 0, 1)
        self.countBox = QLineEdit("0")
        self.countBox.setEnabled(False)
        self.layout().addWidget(self.countBox, 0, 2)
        self.averagesBox = QLineEdit("1")
        self.layout().addWidget(self.averagesBox, 0, 3)

        self.layout().addWidget(QLabel("Frequencies"), 0, 4)
        self.startFreqBox = QLineEdit()
        self.layout().addWidget(self.startFreqBox, 0, 5)
        self.endFreqBox = QLineEdit()
        self.layout().addWidget(self.endFreqBox, 0, 6)

        self.layout().addWidget(QLabel("Constant"), 0, 7)
        self.constBox = QLineEdit()
        self.layout().addWidget(self.constBox, 0, 8)
        self.calibrateBtn = QPushButton("Calibrate")
        self.layout().addWidget(self.calibrateBtn, 0, 9)

class DisplayToolWidget(QWidget):
    def __init__(self):
        super(DisplayToolWidget, self).__init__()

        self.build_widget()

    def build_widget(self):
        self.setLayout(QGridLayout())

        self.openBtn = QPushButton("Open")
        self.layout().addWidget(self.openBtn, 0, 0)
        self.saveBtn = QPushButton("Save")
        self.layout().addWidget(self.saveBtn, 0, 1)

        self.layout().addWidget(QLabel("Frequencies"), 0, 2)
        self.startFreqBox = QLineEdit()
        self.layout().addWidget(self.startFreqBox, 0, 3)
        self.endFreqBox = QLineEdit()
        self.layout().addWidget(self.endFreqBox, 0, 4)

        self.layout().addWidget(QLabel("Constant"), 0, 5)
        self.constBox = QLineEdit()
        self.layout().addWidget(self.constBox, 0, 6)
        self.calibrateBtn = QPushButton("Calibrate")
        self.layout().addWidget(self.calibrateBtn, 0, 7)

class IntegralToolWidget(QWidget):
    def __init__(self):
```

```
        super(IntegralToolWidget, self).__init__()

        self.build_widget()

    def build_widget(self):
        self.setLayout(QGridLayout())

        self.clearBtn = QPushButton("Clear")
        self.layout().addWidget(self.clearBtn, 0, 0)

class Plot(pg.PlotWidget):
    def __init__(self):
        super(Plot, self).__init__()

        self.x = []
        self.y = []

        self.build_widget()

    def build_widget(self):
        self.getPlotItem().getAxis('left').setStyle(tickLength=5)
        self.getPlotItem().getAxis('bottom').setStyle(tickLength=5)
        self.setLabel('bottom', text='m/z')
        self.setLabel('left', text='Intensity')
        self.setDownsampling(auto=True, mode='mean')
        self.setClipToView(True)

    def save_data(self):
        try:
            fileName = QFileDialog.getSaveFileName(filter='Text Files
(*.txt)')[0]
            file = open(fileName, 'w')
            if len(self.x) < len(self.y):
                self.x = range(len(self.y))
            for i in range(len(self.y)):
                file.write(str(self.x[i]))
                file.write("\t")
                file.write(str(self.y[i]))
                file.write("\n")
            file.close()
        except FileNotFoundError:
            None

    def calibrate(self, constant, startFreq, endFreq):
        try:
            startMz = float(constant) / float(startFreq)**2
            endMz = float(constant) / float(endFreq)**2
            stepMz = (endMz - startMz) / len(self.y)
            self.x = [startMz + i * stepMz for i in range(len(self.y))]
        except (ValueError, ZeroDivisionError) as error:
            self.x = range(len(self.y))

        if len(self.x) > 0:
            self.plot(self.x, self.y, clear=True)

class DataPlot(Plot):
    countSignal = pyqtSignal(object)
```

```
    integralSignal = pyqtSignal(object)
    def __init__(self):
        super(DataPlot, self).__init__()

        self.dataSample = 4
        self.numAverages = 1
        self.data = []

        self.build_widget()

    def update(self, data_string):
        self.data.append([data_string[j * 2] + data_string[j * 2 + 1] * 256
for j in range(0, int(len(data_string) / 2), self.dataSample)])
        if abs(len(self.data[-1]) - len(self.y)) > 10 and self.numAverages >
1:
            self.data.pop(-1)
        else:
            if len(self.data) > self.numAverages:
                self.data = self.data[-self.numAverages:]
            self.y = [sum(d) / len(self.data) for d in zip(*self.data)]
            if len(self.y) > 0:
                if len(self.x) != len(self.y):
                    self.x = range(len(self.y))
                self.plot(self.x, self.y, clear=True)
            self.countSignal.emit(str(len(self.data)))
            self.integralSignal.emit(sum(self.data[-1]))

    def set_averages(self, value):
        try:
            self.numAverages = int(value)
        except ValueError:
            self.numAverages = 1

    def set_sample(self, value):
        try:
            self.dataSample = int(value)
        except ValueError:
            self.dataSample = 4

class DisplayPlot(Plot):
    def __init__(self):
        super(DisplayPlot, self).__init__()

        self.build_widget()

    def open_data(self):
        fileName = QFileDialog.getOpenFileName(filter='Text Files
(*.txt)')[0]
        try:
            file = open(fileName, 'r')

            self.x = []
            self.y = []

            for line in file.readlines():
                pair = line.strip('\n').split('\t')
                self.x.append(float(pair[0]))
```

```
                self.y.append(float(pair[1]))
            file.close()

            self.plot(self.x, self.y, clear=True)
        except:
            None

class IntegralPlot(Plot):
    def __init__(self):
        super(IntegralPlot, self).__init__()

        self.data = []

        self.build_widget()

    def build_widget(self):
        super(IntegralPlot, self).build_widget()
        self.setLabel('bottom', text='Time')

    def update(self, point):
        self.data.append(point)
        self.x = range(len(self.data))
        self.y = self.data
        if len(self.y) > 1:
            self.plot(self.x, self.y, clear=True)

    def clr(self):
        self.data = []
```

"DialogWindows.py"

```
from PyQt5.QtWidgets import QDialog, QGridLayout, QLabel, QLineEdit, QFrame,
QPushButton
from PyQt5.QtCore import Qt, pyqtSignal
from SerialPorts import ControlPort, DataPort, DataThread
from serial import SerialException
from math import pi, cos, sin, sqrt, cosh, sinh, acos, inf, floor, log10
import numpy as np
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.figure import Figure


class ConnectionWindow(QDialog):
    def __init__(self, textWidget):
        super(ConnectionWindow, self).__init__()
        self.textWidget = textWidget

        self.controlPort = ControlPort()
        self.dataPort = DataPort(self.controlPort)
        self.dataThread = DataThread(self.controlPort, self.dataPort)

        self.build_window()

        self.signal_handler()
```

```
def build_window(self):
    self.setWindowFlags(Qt.WindowStaysOnTopHint)
    self.setLayout(QGridLayout())

    self.layout().addWidget(QLabel("Control"), 0, 0)
    self.controlBox = QLineEdit("COM10")
    self.layout().addWidget(self.controlBox, 0, 1)

    self.line = QFrame()
    self.line.setFrameShape(QFrame.HLine)
    self.layout().addWidget(self.line, 2, 0, 1, 2)

    self.layout().addWidget(QLabel("Data"), 3, 0)
    self.dataBox = QLineEdit("COM7")
    self.layout().addWidget(self.dataBox, 3, 1)

    self.controlConnectBtn = QPushButton('Connect Controller')
    self.layout().addWidget(self.controlConnectBtn, 1, 0)
    self.controlDisconnectBtn = QPushButton('Disconnect Controller')
    self.layout().addWidget(self.controlDisconnectBtn, 1, 1)
    self.dataConnectBtn = QPushButton('Connect Data')
    self.layout().addWidget(self.dataConnectBtn, 4, 0)
    self.dataDisconnectBtn = QPushButton('Disconnect Data')
    self.layout().addWidget(self.dataDisconnectBtn, 4, 1)

def signal_handler(self):
    self.controlConnectBtn.clicked.connect(self.connect_control)
    self.controlDisconnectBtn.clicked.connect(self.disconnect_control)
    self.dataConnectBtn.clicked.connect(self.connect_data)
    self.dataDisconnectBtn.clicked.connect(self.disconnect_data)

def connect_control(self):
    try:
        self.controlPort.port = self.controlBox.text()
        self.controlPort.open()
        self.dataThread.controlPortAccess = True
        self.dataThread.start()
        self.textWidget.appendPlainText("Controller connected")
    except SerialException:
        self.textWidget.appendPlainText("No serial port found")

def disconnect_control(self):
    try:
        self.dataThread.controlPortAccess = False
        self.controlPort.close()
        self.textWidget.appendPlainText("Controller disconnected")
    except SerialException:
        self.textWidget.appendPlainText("No connection found")

def connect_data(self):
    try:
        self.dataPort.port = self.dataBox.text()
        self.dataPort.open()
        self.dataThread.dataPortAccess = True
        self.textWidget.appendPlainText("Data serial connected")
    except SerialException:
        self.textWidget.appendPlainText("No serial port found")
```

```
    def disconnect_data(self):
        try:
            self.dataThread.dataPortAccess = False
            self.dataPort.close()
            self.textWidget.appendPlainText("Data serial disconnected")
        except SerialException:
            self.textWidget.appendPlainText("No connection found")

class DataSettingsWindow(QDialog):
    def __init__(self):
        super(DataSettingsWindow, self).__init__()
        self.build_window()

    def build_window(self):
        self.setWindowFlags(Qt.WindowStaysOnTopHint)
        self.setLayout(QGridLayout())

        self.layout().addWidget(QLabel("Data point sampling:"), 0, 0)
        self.dataSampleBox = QLineEdit("4")
        self.layout().addWidget(self.dataSampleBox, 0, 1)

        self.applyBtn = QPushButton("Apply")
        self.layout().addWidget(self.applyBtn, 1, 0)

class AddRemoveSegmentWindow(QDialog):
    def __init__(self):
        super(AddRemoveSegmentWindow, self).__init__()
        self.build_window()

    def build_window(self):
        self.setWindowFlags(Qt.WindowStaysOnTopHint)
        self.setLayout(QGridLayout())

        self.layout().addWidget(QLabel("Insert new segment at position:"), 1,
0)
        self.addPositionBox = QLineEdit("1")
        self.layout().addWidget(self.addPositionBox, 1, 1)
        self.layout().addWidget(QLabel("Remove segment from position:"), 3,
0)
        self.removePositionBox = QLineEdit("1")
        self.layout().addWidget(self.removePositionBox, 3, 1)

        self.addSegBtn = QPushButton("Insert segment")
        self.layout().addWidget(self.addSegBtn, 2, 0, 1, 2)
        self.removeSegBtn = QPushButton("Remove segment")
        self.layout().addWidget(self.removeSegBtn, 4, 0, 1, 2)

class CalculatorWindow(QDialog):
    updated = pyqtSignal(object)

    def __init__(self):
        super(CalculatorWindow, self).__init__()

        self.constant = 0

        self.ELECTRON = 1.602e-19
```

```
        self.AMU = 1.66e-27
        self.rBox = QLineEdit("0.707")
        self.zBox = QLineEdit("0.774")
        self.hvBox = QLineEdit("400")
        self.lvBox = QLineEdit("-400")
        self.dBox = QLineEdit("50")
        self.freqBtn = QPushButton("Frequency")
        self.freqBox = QLineEdit("200000")
        self.mzBtn = QPushButton("m/z")
        self.mzBox = QLineEdit("2000")
        self.targetBox = QLineEdit("0.5")
        self.brBox = QLineEdit()
        self.bzBox = QLineEdit()
        self.orBox = QLineEdit()
        self.ozBox = QLineEdit()

        self.build_window()
        self.signal_handler()

    def build_window(self):
        self.setWindowFlags(Qt.WindowStaysOnTopHint)
        self.setLayout(QGridLayout())

        self.layout().addWidget(QLabel("r\u2080 (cm)"), 0, 0)
        self.layout().addWidget(self.rBox, 0, 1)
        self.layout().addWidget(QLabel("z\u2080 (cm)"), 1, 0)
        self.layout().addWidget(self.zBox, 1, 1)

        self.layout().addWidget(QLabel("High V (V)"), 0, 2)
        self.layout().addWidget(self.hvBox, 0, 3)
        self.layout().addWidget(QLabel("Low V (V)"), 1, 2)
        self.layout().addWidget(self.lvBox, 1, 3)
        self.layout().addWidget(QLabel("Duty Cycle (%)"), 2, 2)
        self.layout().addWidget(self.dBox, 2, 3)

        self.layout().addWidget(self.freqBtn, 0, 4)
        self.layout().addWidget(self.freqBox, 0, 5)
        self.layout().addWidget(self.mzBtn, 1, 4)
        self.layout().addWidget(self.mzBox, 1, 5)

        self.layout().addWidget(QLabel("Target Beta z"), 2, 4)
        self.layout().addWidget(self.targetBox, 2, 5)

        self.line = QFrame()
        self.line.setFrameShape(QFrame.HLine)
        self.layout().addWidget(self.line, 3, 0, 1, 6)

        self.layout().addWidget(QLabel("Beta r"), 4, 2)
        self.layout().addWidget(self.brBox, 4, 3)
        self.brBox.setEnabled(False)
        self.layout().addWidget(QLabel("Beta z"), 4, 4)
        self.layout().addWidget(self.bzBox, 4, 5)
        self.bzBox.setEnabled(False)
        self.layout().addWidget(QLabel("omega r (Hz)"), 5, 2)
        self.layout().addWidget(self.orBox, 5, 3)
        self.orBox.setEnabled(False)
        self.layout().addWidget(QLabel("omega z (Hz)"), 5, 4)
```

```
        self.layout().addWidget(self.ozBox, 5, 5)
        self.ozBox.setEnabled(False)

        self.plotBtn = QPushButton("Plot")
        self.layout().addWidget(self.plotBtn, 5, 0)
        self.diagram = Figure()
        self.axes = self.diagram.add_subplot(111)
        self.canvas = FigureCanvasQTAgg(self.diagram)
        self.layout().addWidget(self.canvas, 6, 0, 6, 6)

    def signal_handler(self):
        self.rBox.textChanged.connect(self.update)
        self.zBox.textChanged.connect(self.update)
        self.hvBox.textChanged.connect(self.update)
        self.lvBox.textChanged.connect(self.update)
        self.dBox.textChanged.connect(self.update)
        self.freqBox.textChanged.connect(self.update)
        self.mzBox.textChanged.connect(self.update)

        self.freqBtn.clicked.connect(self.calc_freq)
        self.mzBtn.clicked.connect(self.calc_mz)

        self.plotBtn.clicked.connect(self.updatePlot)

        self.update()
        self.calc_freq()

    def update(self):
        try:
            self.targetBox.setText(str(max(min(float(self.targetBox.text()),
1), 0.0001)))

            duty = float(self.dBox.text())
            freq = float(self.freqBox.text())
            hv = float(self.hvBox.text())
            lv = float(self.lvBox.text())
            mz = float(self.mzBox.text())
            r0 = float(self.rBox.text())
            z0 = float(self.zBox.text())
            betaR = self.calc_beta("r", duty, freq, hv, lv, mz, r0, z0)
            betaZ = self.calc_beta("z", duty, freq, hv, lv, mz, r0, z0)
            self.brBox.setText(str(betaR))
            self.bzBox.setText(str(betaZ))
            self.orBox.setText(str(self.calc_omega(betaR)))
            self.ozBox.setText(str(self.calc_omega(betaZ)))

            self.constant = float(self.freqBox.text())**2 *
float(self.mzBox.text())
            self.updated.emit(str(self.constant))

        except ValueError:
            None

    def calc_M(self, f, d):
        m = [[0 for x in range(2)] for y in range(2)]
        if f > 0:
            m[0][0] = cos(sqrt(f) * d)
```

```
            m[0][1] = 1 / sqrt(f) * sin(sqrt(f) * d)
            m[1][0] = -sqrt(f) * sin(sqrt(f) * d)
            m[1][1] = cos(sqrt(f) * d)
        else:
            m[0][0] = cosh(sqrt(-f) * d)
            m[0][1] = 1 / sqrt(-f) * sinh(sqrt(-f) * d)
            m[1][0] = sqrt(-f) * sinh(sqrt(-f) * d)
            m[1][1] = cosh(sqrt(-f) * d)
        return(m)

    def calc_beta(self, dim, duty, freq, hv, lv, mz, r0, z0):
        try:
            if dim in ["x", "z"]:
                c = -8
            elif dim == "y":
                c = 8
            elif dim == "r":
                c = 4
            else:
                return(ValueError)

            fHi = 2 * c * self.ELECTRON * hv / (mz * self.AMU) / pow(freq * 2
* pi, 2) / (pow(r0 / 100, 2) + 2* pow(z0 / 100, 2))
            dHi = duty / 100 * pi
            mHi = self.calc_M(fHi, dHi)

            fLo = 2 * c * self.ELECTRON * lv / (mz * self.AMU) / pow(freq * 2
* pi, 2) / (pow(r0 / 100, 2) + 2 * pow(z0 / 100, 2))
            dLo = (1 - duty / 100) * pi
            mLo = self.calc_M(fLo, dLo)

            m = np.dot(mHi, mLo)

            beta = acos((m[0][0] + m[1][1]) / 2) / pi

        except:
            beta = inf

        return(beta)

    def calc_omega(self, beta):
        try:
            omega = 1/2 * beta * float(self.freqBox.text())

        except:
            omega = inf

        return(omega)

    def calc_freq(self):
        try:
            float(self.mzBox.text())
            for i in range(floor(log10(float(self.freqBox.text()))), -1, -1):
                if float(self.brBox.text()) == inf or
float(self.bzBox.text()) == inf:
                    None
                elif float(self.bzBox.text()) < float(self.targetBox.text()):
```

```
                    while float(self.bzBox.text()) <
float(self.targetBox.text()) and float(self.brBox.text()) < inf and
float(self.freqBox.text()) > 0:
                        self.freqBox.setText(str(float(self.freqBox.text()) -
pow(10, i)))
                    self.freqBox.setText(str(float(self.freqBox.text()) +
pow(10, i)))
                else:
                    while float(self.bzBox.text()) >
float(self.targetBox.text()) and float(self.bzBox.text()) < inf and
float(self.brBox.text()) < inf and float(self.freqBox.text()) > 0:
                        self.freqBox.setText(str(float(self.freqBox.text()) +
pow(10, i)))
                    self.freqBox.setText(str(float(self.freqBox.text()) -
pow(10, i)))
        except ValueError:
            None

    def calc_mz(self):
        try:
            float(self.freqBox.text())
            for i in range(floor(log10(float(self.mzBox.text()))), -1, -1):
                if float(self.brBox.text()) == inf or
float(self.bzBox.text()) == inf:
                    None
                elif float(self.bzBox.text()) < float(self.targetBox.text()):
                    while float(self.bzBox.text()) <
float(self.targetBox.text()) and float(self.brBox.text()) < inf and
float(self.mzBox.text()) > 0:
                        self.mzBox.setText(str(float(self.mzBox.text()) -
pow(10, i)))
                    self.mzBox.setText(str(float(self.mzBox.text()) + pow(10,
i)))
                else:
                    while float(self.bzBox.text()) >
float(self.targetBox.text()) and float(self.bzBox.text()) < inf and
float(self.brBox.text()) < inf and float(self.mzBox.text()) > 0:
                        self.mzBox.setText(str(float(self.mzBox.text()) +
pow(10, i)))
                    self.mzBox.setText(str(float(self.mzBox.text()) - pow(10,
i)))
        except ValueError:
            None

    def updatePlot(self):
        try:
            hv = float(self.hvBox.text())
            lv = float(self.lvBox.text())
            mz = float(self.mzBox.text())
            r0 = float(self.rBox.text())
            z0 = float(self.zBox.text())

            f = float(self.freqBox.text())
            q = 2 * 4 * self.ELECTRON * (hv - lv) / 2 / (mz * self.AMU) /
pow(f * 2 * pi, 2) / (pow(r0 / 100, 2) + 2* pow(z0 / 100, 2))
            d = float(self.dBox.text())
```

```
            plotQ = np.arange(0.0001, max(1, q + 0.1), max(1, q + 0.1) / 200)
            plotF = np.sqrt(2 * 4 * self.ELECTRON * (hv - lv) / 2 / (mz *
self.AMU) / plotQ / (pow(r0 / 100, 2) + 2* pow(z0 / 100, 2))) / (2 * pi)
            plotD = np.arange(min(40, d - 5), max(70, d + 5), (max(70, d + 5)
- min(40, d - 5)) / 150)

            plotBR = np.array([[self.calc_beta("r", d, f, hv, lv, mz, r0, z0)
for f in plotF] for d in plotD])
            plotBZ = np.array([[self.calc_beta("z", d, f, hv, lv, mz, r0, z0)
for f in plotF] for d in plotD])
            plotB = 255 - np.dstack((np.logical_and(plotBR != inf, plotBZ ==
inf), np.logical_and(plotBR != inf, plotBZ != inf), np.logical_and(plotBR ==
inf, plotBZ != inf))) * 255

            self.axes.clear()
            self.axes.imshow(plotB, aspect='auto', origin='lower',
extent=[min(plotQ), max(plotQ), min(plotD), max(plotD)])
            self.axes.set_xticks(plotQ[0::20])
            self.axes.set_xticklabels(np.around(plotF[0::20] / 1000))
            self.axes.set_xlabel("Fequency (kHz)")
            self.axes.set_yticks(plotD[0::20])
            self.axes.set_ylabel("Duty Cycle (%)")
            self.axes.set_title("Stability for m/z " + self.mzBox.text())
            self.axes.plot(q, d, marker='o', color='black')
            self.canvas.draw()
        except ValueError:
            None
```

“SerialPorts.py”

```
from PyQt5.QtCore import QThread, pyqtSignal
from serial import Serial, SerialException


class ControlPort(Serial):
    def __init__(self):
        super(ControlPort, self).__init__(baudrate=2000000, timeout=5)

    def serial_write(self, output):
        self.write(output.encode('ascii'))

    def serial_read(self, stopString):
        readInput = []
        try:
            readInput.append(self.readline().decode('ascii').strip())
            while readInput[-1] != stopString and readInput[-1] != "":
                readInput.append(self.readline().decode('ascii').strip())
        except SerialException:
            readInput.append("Serial read failed")
        return readInput

class DataPort(Serial):
    def __init__(self, controlPort):
        super(DataPort, self).__init__()
```

```
class DataThread(QThread):
    dataSignal = pyqtSignal(object)
    textSignal = pyqtSignal(object)

    def __init__(self, controlPort, dataPort):
        super(DataThread, self).__init__()
        self.daemon = True

        self.dataPort = dataPort
        self.controlPort = controlPort

        self.n = 0
        self.maxNumData = 100
        self.dataString = [b'' for n in range(self.maxNumData)]

        self.controlPortAccess = False
        self.dataPortAccess = False

    def run(self):
        while self.controlPortAccess:
            if self.controlPort.in_waiting:

self.textSignal.emit(self.controlPort.read(self.controlPort.in_waiting).decod
e('ascii').strip())
                if self.dataPortAccess:
                    while self.dataPort.in_waiting:
                        self.dataString[self.n] +=
self.dataPort.read(self.dataPort.in_waiting)

self.dataSignal.emit(self.dataString[self.n].strip(b'stop'))
                    self.n = (self.n + 1) % self.maxNumData
                    self.dataString[self.n] = b''
                    self.dataPort.reset_input_buffer()
            if self.dataPortAccess:
                if self.dataPort.in_waiting:
                    self.dataString[self.n] +=
self.dataPort.read(self.dataPort.in_waiting)
```

“ScanFunction.py”

```
from PyQt5.QtWidgets import QSplitter, QMessageBox, QFileDialog, QScrollArea,
QWidget, QGridLayout, QTableView
from PyQt5.QtCore import Qt, QAbstractTableModel, QModelIndex
from PyQt5.QtGui import QBrush, QColor
from collections import OrderedDict
import json


class ScanFunction(list):
    def __init__(self):
        super(ScanFunction, self).__init__()

    def reset(self, newScanFunction):
        while len(self) > 0:
            self.remove(self[0])
```

```
        for seg in newScanFunction:
            self.append(seg)

class ScanWidget(QSplitter):
    def __init__(self, textWidget):
        super(ScanWidget, self).__init__(Qt.Vertical)

        self.textWidget = textWidget
        self.scanFunction = ScanFunction()

        self.scanArea = ScanArea(self.scanFunction)

        self.build_widget()

    def build_widget(self):
        self.addWidget(self.scanArea)

    def save_check(self):
        msgBox = QMessageBox()
        msgBox.setText("Do you want to save the current scan?")
        msgBox.setStandardButtons(QMessageBox.Yes | QMessageBox.No |
QMessageBox.Cancel)
        return msgBox.exec()

    def save_scan(self):
        try:
            fileName = QFileDialog.getSaveFileName(filter='Scan Files
(*.scan);;Text Files (*.txt)')[0]
            file = open(fileName, 'w')
            file.write(json.dumps(self.scanFunction, sort_keys=False,
indent=4))
            file.close()

            scanPic = self.scanArea.widget().grab()
            scanPicFileName = fileName.replace('.scan', '.jpg')
            scanPic.save(scanPicFileName, 'jpg')
        except FileNotFoundError:
            self.textWidget.appendPlainText("File save failed")

    def open_scan(self):
        choice = self.save_check()
        if choice == QMessageBox.Cancel:
            pass
        else:
            if choice == QMessageBox.Yes:
                self.save_scan()

            fileName = QFileDialog.getOpenFileName(filter='Scan Files
(*.scan);;Text Files (*.txt);;All Files (*.*)')[0]

            try:
                file = open(fileName, 'r')
                scanData = file.read()

                newScanFunction = json.loads(scanData)
                self.scanArea.reset(len(newScanFunction))
                self.scanFunction.reset(newScanFunction)
```

```
                file.close()
            except FileNotFoundError:
                self.textWidget.appendPlainText("File open failed")

class ScanArea(QScrollArea):
    def __init__(self, scanFunction):
        super(ScanArea, self).__init__()

        self.headerLabels = ["Name", "Active", "Record", "Duration"]
        self.headerTypes = [str, Bool, Bool, PosInt]
        self.OUTPUTS = 3
        self.outputLabels = ["Start", "End", "Duty Cycle", "Tickle",
"Amplitude", "Phase"]
        self.outputTypes = [FreqFloat, FreqFloat, DCFloat, DivChoice,
AmpFloat, PhaseChoice]
        self.ANALOG = 8
        self.analogLabels = ["A" + str(i + 1) for i in range(self.ANALOG)]
        self.analogTypes = [RangedFloat for i in range(self.ANALOG)]
        self.DIGITAL = 12
        self.digitalLabels = ["D" + str(i + 1) for i in range(self.DIGITAL)]
        self.digitalTypes = [Bool for i in range(self.DIGITAL)]

        self.build_widget(scanFunction)

    def build_widget(self, scanFunction):
        self.setWidget(QWidget())
        self.setWidgetResizable(True)
        self.widget().setLayout(QGridLayout())
        self.widget().layout().setAlignment(Qt.AlignLeft)

        self.headerView = ParameterView()
        self.headerView.setupModel(HeaderModel(scanFunction,
self.headerLabels, self.headerTypes))

self.headerView.model().dataChanged.connect(self.headerView.viewport().update
)
        self.outputViewList = []
        for i in range(self.OUTPUTS):
            self.outputViewList.append(ParameterView())
            self.outputViewList[i].setupModel(OutputModel(scanFunction, i,
self.outputLabels, self.outputTypes))

self.headerView.model().dataChanged.connect(self.outputViewList[i].viewport()
.update)
        self.analogView = ParameterView()
        self.analogView.setupModel(ListParsModel(scanFunction, "Analog",
self.analogLabels, self.analogTypes))

self.headerView.model().dataChanged.connect(self.analogView.viewport().update
)
        self.digitalView = ParameterView()
        self.digitalView.setupModel(ListParsModel(scanFunction, "Digital",
self.digitalLabels, self.digitalTypes))

self.headerView.model().dataChanged.connect(self.digitalView.viewport().updat
e)
```

```
        self.widget().layout().addWidget(self.headerView)
        for i in range(self.OUTPUTS):
            self.widget().layout().addWidget(self.outputViewList[i])
        self.widget().layout().addWidget(self.analogView)
        self.widget().layout().addWidget(self.digitalView)

    def reset(self, newLength):
        while self.remove_segment(0):
            None
        for i in range(newLength):
            self.add_segment(i)

    def add_segment(self, position):
        self.headerView.model().insertColumn(position, QModelIndex())
        for i in range(self.OUTPUTS):
            self.outputViewList[i].model().insertColumn(position,
QModelIndex())
        self.analogView.model().insertColumn(position, QModelIndex())
        self.digitalView.model().insertColumn(position, QModelIndex())

    def remove_segment(self, position):
        if self.headerView.model().removeColumn(position, QModelIndex()):
            for i in range(self.OUTPUTS):
                self.outputViewList[i].model().removeColumn(position,
QModelIndex())
            self.analogView.model().removeColumn(position, QModelIndex())
            self.digitalView.model().removeColumn(position, QModelIndex())
            return True
        else:
            return False

    def mousePressEvent(self, event):
        self.headerView.clearSelection()
        for i in range(self.OUTPUTS):
            self.outputViewList[i].clearSelection()
        self.analogView.clearSelection()
        self.digitalView.clearSelection()

class ParameterView(QTableView):
    def __init__(self):
        super(ParameterView, self).__init__()

    def setupModel(self, model):
        self.setModel(model)
        self.verticalHeader().setFixedWidth(100)
        self.setFixedSize(self.horizontalHeader().length() +
self.verticalHeader().width() + 2, self.horizontalHeader().height() +
self.verticalHeader().length() + 2)
        self.model().columnsInserted.connect(lambda:
self.setFixedSize(self.horizontalHeader().length() +
self.verticalHeader().width() + 2, self.horizontalHeader().height() +
self.verticalHeader().length() + 2))
        self.model().columnsRemoved.connect(lambda:
self.setFixedSize(self.horizontalHeader().length() +
self.verticalHeader().width() + 2, self.horizontalHeader().height() +
self.verticalHeader().length() + 2))
```

```
class ParameterModel(QAbstractTableModel):
    def __init__(self, scanFunction, labels, types):
        super(ParameterModel, self).__init__()

        self.scanFunction = scanFunction

        self.labels = labels
        self.types = types

    def rowCount(self, parent):
        return len(self.labels)

    def columnCount(self, parent):
        return(len(self.scanFunction))

    def headerData(self, section, orientation, role):
        if role == Qt.DisplayRole:
            if orientation == Qt.Vertical:
                return self.labels[section]
            else:
                return section + 1

    def flags(self, index):
        return Qt.ItemIsEditable | QAbstractTableModel.flags(self, index)

class HeaderModel(ParameterModel):
    def __init__(self, scanFunction, labels, types):
        super(HeaderModel, self).__init__(scanFunction, labels, types)

    def data(self, index, role):
        if role in [Qt.DisplayRole, Qt.EditRole]:
            return
str(self.scanFunction[index.column()][self.labels[index.row()]])
        if role == Qt.BackgroundRole:
            if self.scanFunction[index.column()]["Active"] == "False":
                return QBrush(QColor('grey'))
            else:
                if
self.scanFunction[index.column()][self.labels[index.row()]] == "False":
                    return QBrush(QColor('red'))
                elif
self.scanFunction[index.column()][self.labels[index.row()]] == "True":
                    return QBrush(QColor('green'))
                else:
                    return QBrush(QColor('white'))

    def setData(self, index, value, role):
        if role == Qt.EditRole:
            self.scanFunction[index.column()][self.labels[index.row()]] =
self.types[index.row()](value)
            self.dataChanged.emit(index, index)
        return True

    def insertColumn(self, position, parent):
        if position <= len(self.scanFunction) and position >= 0:
            self.beginInsertColumns(parent, position, position)
```

```
            self.scanFunction.insert(position, OrderedDict())
            for label, type in zip(self.labels, self.types):
                if position > 0:
                    self.scanFunction[position][label] =
self.scanFunction[position - 1][label]
                else:
                    self.scanFunction[position][label] = type(0)
            self.scanFunction[position]["Outputs"] = []
            self.endInsertColumns()
            return True
        else:
            return False

    def removeColumn(self, position, parent):
        if position < len(self.scanFunction) and position >= 0:
            self.beginRemoveColumns(parent, position, position)
            self.scanFunction.remove(self.scanFunction[position])
            self.endRemoveColumns()
            return True
        else:
            return False

class OutputModel(ParameterModel):
    def __init__(self, scanFunction, output, labels, types):
        super(OutputModel, self).__init__(scanFunction, labels, types)

        self.output = output

    def data(self, index, role):
        if role in [Qt.DisplayRole, Qt.EditRole]:
            return
str(self.scanFunction[index.column()]["Outputs"][self.output][self.labels[ind
ex.row()]])
        if role == Qt.BackgroundRole:
            if self.scanFunction[index.column()]["Active"] == "False":
                return QBrush(QColor('grey'))
            else:
                return QBrush(QColor('white'))
        if role == Qt.ToolTipRole:
            if self.labels[index.row()] == "Tickle":
                return "Div / [value here] or Output 3"
            if self.labels[index.row()] == "Phase":
                return "0 or 180 (degrees)"

    def setData(self, index, value, role):
        if role == Qt.EditRole:

self.scanFunction[index.column()]["Outputs"][self.output][self.labels[index.r
ow()]] = self.types[index.row()](value)
            self.dataChanged.emit(index, index)
        return True

    def insertColumn(self, position, parent):
        if position <= len(self.scanFunction) and position >= 0:
            self.beginInsertColumns(parent, position, position)
            self.scanFunction[position]["Outputs"].append(OrderedDict())
            for label, type in zip(self.labels, self.types):
```

```
                if position > 0:

self.scanFunction[position]["Outputs"][self.output][label] =
self.scanFunction[position - 1]["Outputs"][self.output][label]
                else:

self.scanFunction[position]["Outputs"][self.output][label] = type(0)
            self.endInsertColumns()
            return True
        else:
            return False

    def removeColumn(self, position, parent):
        self.beginRemoveColumns(parent, position, position)
        self.endRemoveColumns()

class ListParsModel(ParameterModel):
    def __init__(self, scanFunction, name, labels, types):
        super(ListParsModel, self).__init__(scanFunction, labels, types)

        self.name = name

    def data(self, index, role):
        if role in [Qt.DisplayRole, Qt.EditRole]:
            return
str(self.scanFunction[index.column()][self.name][index.row()])
        if role == Qt.BackgroundRole:
            if self.scanFunction[index.column()]["Active"] == "False":
                return QBrush(QColor('grey'))
            else:
                if self.scanFunction[index.column()][self.name][index.row()]
== "False":
                    return QBrush(QColor('red'))
                elif
self.scanFunction[index.column()][self.name][index.row()] == "True":
                    return QBrush(QColor('green'))
                else:
                    return QBrush(QColor('white'))

    def setData(self, index, value, role):
        if role == Qt.EditRole:
            self.scanFunction[index.column()][self.name][index.row()] =
self.types[index.row()](value)
            self.dataChanged.emit(index, index)
        return True

    def insertColumn(self, position, parent):
        if position <= len(self.scanFunction) and position >= 0:
            self.beginInsertColumns(parent, position, position)
            self.scanFunction[position][self.name] = []
            for i in range(len(self.labels)):
                if position > 0:

self.scanFunction[position][self.name].append(self.scanFunction[position -
1][self.name][i])
                else:
```

```
self.scanFunction[position][self.name].append(self.types[i](0))
            self.endInsertColumns()
            return True
        else:
            return False

    def removeColumn(self, position, parent):
        self.beginRemoveColumns(parent, position, position)
        self.endRemoveColumns()

class RangedFloat(float):
    def __new__(cls, value):
        try:
            v = float.__new__(cls, min(max(-10, float(value)), 10))
        except ValueError:
            v = 0
        return v

class FreqFloat(float):
    def __new__(cls, value):
        try:
            v = float.__new__(cls, min(max(0, float(value)), 1000000))
        except ValueError:
            v = 0
        return v

class DCFloat(float):
    def __new__(cls, value):
        try:
            v = float.__new__(cls, min(max(0, float(value)), 100))
        except ValueError:
            v = 0
        return v

class AmpFloat(float):
    def __new__(cls, value):
        try:
            v = float.__new__(cls, min(max(0, float(value)), 5))
        except ValueError:
            v = 0
        return v

class PosInt(int):
    def __new__(cls, value):
        try:
            v = int.__new__(cls, max(0, round(float(value))))
        except ValueError:
            v = 0
        return v

class Bool(int):
    def __new__(cls, value):
        try:
            s = int.__new__(cls, bool(int(value)))
        except ValueError:
            s = 0
```

```
        l = ["False", "True"]
        return l[s]

class DivChoice(str):
    def __new__(cls, value):
        try:
            v = int(value)
            if v == 2 or v == 4 or v == 8 or v == 16:
                s = str.__new__(cls, "Div / " + str(v))
            else:
                raise ValueError()
        except ValueError:
            s = "Output 3"
        return s

class PhaseChoice(int):
    def __new__(cls, value):
        try:
            v = int(value)
            if v == 0:
                s = v
            else:
                s = 180
        except ValueError:
            s = 0
        return s
```

APPENDIX C. R SHINY APPLICATION CODE

"ui.R"

```
library(shiny)            # Include Shiny App library
library(rhandsontable)    # Library for interactive tables
library(plotly)           # Library for interactive plots


# Defines tab UI for each ion polarity
#   id = identifier to call object in app
#   name = display name in app
ionUI <- function(id, name){
  ns <- NS(id)        # Puts id in same namespace as app.
  tabPanel(name,

           fluidRow(
             column(6, actionButton(ns("add"), "Add Component")),
             column(6, actionButton(ns("rem"), "Remove Last"))
           ),

           tabsetPanel(id = ns("ion")))
}


# Defines plot UI
plotUI <- function(id, name){
  ns <- NS(id)
  tabPanel(name,

           fluidRow(
             column(6, selectInput(ns("var"), "Plot Variable", NULL)),
             column(6, numericInput(ns("pointRes"), "m/z per Data Point",
0.01))
           ),

           plotlyOutput(ns("plot")),

           fluidRow(
             column(6, fileInput(ns("openData"), "Open Data")),
             column(6, downloadButton(ns("saveData"), "Save Model Data"))
           ))

}


# Defines main app UI elements
shinyUI(fluidPage(

  titlePanel("Ion/ion Reaction Calculator"),

  sidebarLayout(

    sidebarPanel(
```

```
      fileInput("open", "Open Parameters File"),
      downloadButton("save", "Save Parameters File"),

      numericInput("peakRes", "Min Peak FWHM (m/z)", 0.1),
      actionButton("calc", "Calculate and Plot"),


      tabsetPanel(
        id = "pars",

        ionUI("pos", "Positive"),          # Calls module for positive ion UI.
        ionUI("neg", "Negative")          # Calls module for negative ion UI.

      )

    ),

    mainPanel(

      tabsetPanel(

        plotUI("pos", "Positive"),
        plotUI("neg", "Negative"),
        plotUI("rxn", "Reaction")

      )

    )

  )

))
```

“server.R”

```
library(shiny)
library(jsonlite)     # Library for reading and writing parameter files
library(reshape2)     # Library for handling data structures in calculations
library(dplyr)        # Library for data manipulation

source("SpecCalc.R")
source("VarsCalc.R")
source("DataCalc.R")
source("Plot.R")
source("model.R")

options(shiny.maxRequestSize = 30*1024^2)


# Convert table values to data structure for calculations
hotToR <- function(hot){
  hot_data <- lapply(hot$data, function(dr) lapply(dr, function(dc)
ifelse(is.null(dc), NA, dc)))
  df <- as.data.frame(t(sapply(hot_data, unlist)))
```

```
  as.list(setNames(df, hot$params$colHeaders))
}


# Defines tab UI for individual components
#   id = identifier used in app
#   n = component number, displayed in app
compUI <- function(id, n){
  ns <- NS(id)
  tabPanel(n(),

           textOutput(ns("base.mass.title")),
           rHandsontableOutput(ns("base.mass"), 200),

           textOutput(ns("unit.mass.title")),
           rHandsontableOutput(ns("unit.mass"), 200),

           textOutput(ns("unit.num.title")),
           rHandsontableOutput(ns("unit.num"), 200),

           textOutput(ns("add.title")),
           rHandsontableOutput(ns("add"), 200),
           textOutput(ns("add.ratio.title")),
           rHandsontableOutput(ns("add.ratio"), 200))
}

# Module defining update of component tab displays.
#   input, output, session = required by Shiny framework
#   pars = reactive input parameters, module display updates if pars changes
compMod <- function(input, output, session, pars){

  parameters <- req(pars())     # Module won't run until pars returns a
"truthy" value

  # suspendWhenHidden set to FALSE causes the app to render objects in the
background
  output$base.mass.title <- renderText("Non-repeating Mass Distribution")
  outputOptions(output, "base.mass.title", suspendWhenHidden = FALSE)

  output$base.mass <- renderRHandsontable(
    with(parameters$analyte_pars$base_pars,
         rhandsontable(data.frame(mass = mass, abund = abund, width = width),
rowHeaders = NULL))
  )
  outputOptions(output, "base.mass", suspendWhenHidden = FALSE)

  output$unit.mass.title <- renderText("Repeating Mass Distribution")
  outputOptions(output, "unit.mass.title", suspendWhenHidden = FALSE)

  output$unit.mass <- renderRHandsontable(
    with(parameters$analyte_pars$unit_pars,
         rhandsontable(data.frame(mass = mass, abund = abund, width = width),
rowHeaders = NULL))
  )
  outputOptions(output, "unit.mass", suspendWhenHidden = FALSE)

  output$unit.num.title <- renderText("Repeating Unit Number Range")
```

```
  outputOptions(output, "unit.num.title", suspendWhenHidden = FALSE)

  output$unit.num <- renderRHandsontable(
    with(parameters$analyte_pars$number_pars,
         rhandsontable(data.frame(low = low, high = high, mode = mode,
entropy = entropy), rowHeaders = NULL))
  )
  outputOptions(output, "unit.num", suspendWhenHidden = FALSE)

  output$add.title <- renderText("Definition of Charged Adduct")
  outputOptions(output, "add.title", suspendWhenHidden = FALSE)

  output$add <- renderRHandsontable(
    with(parameters$agent_pars,
         rhandsontable(data.frame(mass = mass, charge = charge), rowHeaders =
NULL))
  )
  outputOptions(output, "add", suspendWhenHidden = FALSE)

  output$add.ratio.title <- renderText("Adduct-to-Number Ratio Range")
  outputOptions(output, "add.ratio.title", suspendWhenHidden = FALSE)

  output$add.ratio <- renderRHandsontable(
    with(parameters$agent_pars$ratio_pars,
         rhandsontable(data.frame(low = low, high = high, mode = mode,
entropy = entropy), rowHeaders = NULL))
  )
  outputOptions(output, "add.ratio", suspendWhenHidden = FALSE)

}


# Module defining display and behavior of ion polarity tabs.
#   inList = reactive list of components and parameters, module updates as
list changes
ionMod <- function(input, output, session, inList){

  ns <- session$ns       # Used to keep all objects in same namespace as main
app
  n <- reactiveVal(0)    # reactive - current number of components

  # Function to add a component on button click.
  # n is updated, tab is added to UI, new module is called for added
component
  observeEvent(input$add, {

    n(n() + 1)

    appendTab("ion", compUI(ns(paste(n())), n))
    callModule(compMod, paste(n()), reactive(model))

  })

  # Function to remove last added component
  # n is updated, tab is removed
  # technically last component underlying object still exists but cannot be
accessed
```

```
  observeEvent(input$rem, {

    if(n() > 0){
      removeTab("ion", paste(n()))
      n(n() - 1)
    }

  })

  # Observes changes in inList.
  # Clears all current component tabs, adds new tabs based on inList and
updates n
  observe({

    for(i in 1:isolate(n())) removeTab("ion", paste(i))
    n(0)

    if(length(inList()) > 0){

      for(i in 1:length(inList())){

        appendTab("ion", compUI(ns(i), reactive(i)))
        callModule(compMod, paste(i), reactive(inList()[[i]]))

      }

      n(length(inList()))

    }

  })

  # Reload module when tab is clicked.
  # Purely asthetic, tables in component tabs don't load until clicked on
without this.
  observeEvent(input$ion, callModule(compMod, input$ion,
reactive(outList()[[input$ion]])))

  # Reactive list of components and parameters to calculate plots.
  # Updates when tables in component tabs are modified.
  # Return value of module.
  outList <- reactive({
    temp <- list()
    if(n() > 0){
      req(input[[paste0(n(), "-base.mass")]])   # Ensures that tables exist
      for(comp in 1:n()){
        temp[[paste(comp)]] <- list(
          analyte_pars = list(
            base_pars = hotToR(input[[paste0(comp, "-base.mass")]]),
            unit_pars = hotToR(input[[paste0(comp, "-unit.mass")]]),
            number_pars = hotToR(input[[paste0(comp, "-unit.num")]])
          ),
          agent_pars = c(
            hotToR(input[[paste0(comp, "-add")]]),
            list(ratio_pars = hotToR(input[[paste0(comp, "-add.ratio")]]))
          )
        )
```

```
      }
    }
    temp
  })

  return(outList)

}

# Module for plot tabs
plotMod <- function(input, output, session, plot_data){
  # Generate and update plots when new data calculated
  observe({
    iplt <- iPlotly(plot_data(), input$var, file_data())
    output$plot <- renderPlotly(iplt)
  })

  # Update plot variable choices
  observe(updateSelectInput(session, "var", NULL,
names(plot_data()$indiv[!names(plot_data()$indiv) %in% c("i", "x", "y")])))

  # Open file data, causes plot to update
  file_data <- reactive({

    file <- input$openData
    if(!is.null(file)){
      d <- as_tibble(read.table(file$datapath, col.names = c('x', 'y')))
      d$y <- d$y / max(d$y) * 100
      d
    }

  })

  # Save model data
  output$saveData <- downloadHandler(
    filename = paste0(Sys.Date(), ".txt"),
    content = function(file) write.table(plot_data()$indiv %>% dcast(x ~
get(input$var), value.var = 'y', fun.aggregate = sum),
                                         file, sep = "\t",
                                         row.names = FALSE, col.names = TRUE)
  )

  return(reactive(input$pointRes))

}



# Main server for app
shinyServer(function(input, output, session) {

  # Reactive lists holding components and parameters, returned by modules
  # Update when inPars changes
  posList <- callModule(ionMod, "pos", reactive(inPars()$pos))
  negList <- callModule(ionMod, "neg", reactive(inPars()$neg))

  # Reactive lists holding plot parameters, returned by modules
```

```
  posPointRes <- callModule(plotMod, "pos", reactive(plot_data()$positive))
  negPointRes <- callModule(plotMod, "neg", reactive(plot_data()$negative))
  rxnPointRes <- callModule(plotMod, "rxn", reactive(plot_data()$reaction))

  # Calculate data for the plots - triggered by the Calculate button
  # Returns list of data for plots.
  plot_data <- eventReactive(input$calc,

    calcData(posList(), negList(), input$peakRes / 2.355,
             posPointRes(), negPointRes(), rxnPointRes())

  )

  # Open parameters file. Triggered by opening a file.
  inPars <- reactive({

    if(isTruthy(input$open)){
      inFile <- input$open
      temp_pars <- read_json(inFile$datapath, TRUE)
    }
    else
      temp_pars <- NULL
    list(pos = temp_pars$positive, neg = temp_pars$negative, rxn =
temp_pars$reaction)

  })

  # Save parameters file
  outPars <- reactive(list(positive = posList(), negative = negList()))
  output$save <- downloadHandler(
    filename = paste0("pars_", Sys.Date(), ".txt"),
    content = function(file) write_json(outPars(), file, pretty = TRUE)
  )

})
```

“SpecCalc.R”

```
# Calculate data used in plots based on positive and negative ion
definitions.
calcData <- function(positive = NULL, negative = NULL, peak_resolution,
                     pos_point_res, neg_point_res, rxn_point_res){

  plot_data <- list(positive = NULL, negative = NULL, complex = NULL)

  # Postive Ion
  if(length(positive) > 0){

    pos_data <- melt(lapply(positive, totalVarsCalc))

    names(pos_data)[length(names(pos_data)) - 2:0] <- c("num", "names",
"comp")
    pos_data$comp <- as.integer(pos_data$comp)
    pos_data$num <- as.integer(pos_data$num)
    pos_data <- dcast(pos_data, comp + num + mass + charge ~ names)
```

```
    pos_data <- pos_data %>% mutate(mz = m / z, abund = h, width = w / z)

    plot_data$positive <- plotData(pos_data %>% select(-c("m", "z", "h",
"w")),
                                   pos_point_res, peak_resolution)

  }

  # Negative Ion
  if(length(negative) > 0){

    neg_data <- melt(lapply(negative, totalVarsCalc))

    names(neg_data)[length(names(neg_data)) - 2:0] <- c("num", "names",
"comp")
    neg_data$comp <- as.integer(neg_data$comp)
    neg_data$num <- as.integer(neg_data$num)
    neg_data <- dcast(neg_data, comp + num + mass + charge ~ names)

    neg_data <- neg_data %>% mutate(mz = m / z, abund = h, width = w / z)

    plot_data$negative <- plotData(neg_data %>% select(-c("m", "z", "h",
"w")),
                                   neg_point_res, peak_resolution)

  }

  # Reaction
  if(length(positive) > 0 & length(negative) > 0){

    names(pos_data) <- paste0("pos_", names(pos_data))
    names(neg_data) <- paste0("neg_", names(neg_data))

    rxn_data <- merge(pos_data, neg_data) %>%
      select(-c("pos_mz", "neg_mz", "pos_abund", "neg_abund", "pos_width",
"neg_width")) %>%
      mutate(m = pos_m + neg_m,
             z = pos_z + neg_z) %>%
      select(-c("pos_m", "neg_m", "pos_z", "neg_z")) %>%
      mutate(mz = m / z,
             abund = pos_h * neg_h,
             width = sqrt(pos_w^2 + neg_w^2) / z) %>%
      select(-c("m", "z", "pos_h", "neg_h", "pos_w", "neg_w"))

    plot_data$reaction <- plotData(rxn_data, rxn_point_res, peak_resolution)

  }

  plot_data

}
```

“VarsCalc.R”

```
gauss <- function(mu, h, sig, x){
  if(sig == 0 | is.na(sig)) sig <- 1
  h / sig / sqrt(2*pi) * exp(-(x - mu)^2 / 2 / sig^2)
}

ngamma <- function(x, shape, rate){
  p <- dgamma(x, shape, rate)
  p / sum(p)
}

eround <- function(x, end){
  if(x %% 0.5 == 0){
    if(end == "low") floor(x)
    else ceiling(x)
  }
  else round(x)
}

dstrCalc <- function(base_pars, unit_pars, number_pars){

  final_dstr <- list()
  exp_dstr <- data.frame(mass = base_pars$mass, abund = base_pars$abund,
width = base_pars$width)

  num_dstr <- with(
    number_pars,
    # data.frame(num = low:high, abund = gauss(mu, 1, sig, low:high))
    data.frame(num = low:high, abund = dgamma(low:high+1, (mode+1)/entropy+1,
1/entropy))
  )

  if(0 %in% num_dstr$num){

    final_dstr[[length(final_dstr) + 1]] <- exp_dstr
    final_dstr[[length(final_dstr)]]$abund <-
final_dstr[[length(final_dstr)]]$abund * num_dstr$abund[num_dstr$num == 0]

  }

  for(i in 1:max(num_dstr$num)){

    exp_mass <- outer(exp_dstr$mass, unit_pars$mass, FUN = "+")
    exp_abund <- outer(exp_dstr$abund, unit_pars$abund)
    exp_width <- sqrt(outer(exp_dstr$width^2, unit_pars$width^2, FUN = "+"))

    exp_dstr <- data.frame(mass = as.vector(round(exp_mass, 6)), abund =
as.vector(exp_abund), width = as.vector(exp_width)) %>%
      group_by(mass) %>% summarize(abund = sum(abund), width =
sqrt(sum(width^2)))

    if(i %in% num_dstr$num){

      final_dstr[[length(final_dstr) + 1]] <- exp_dstr
```

```
      final_dstr[[length(final_dstr)]]$abund <-
final_dstr[[length(final_dstr)]]$abund * num_dstr$abund[num_dstr$num == i]

    }

  }

  final_dstr

}

totalVarsCalc <- function(pars){

  analyte_dstr <- with(pars$analyte_pars, dstrCalc(base_pars, unit_pars,
number_pars))

  agent_dstr <- lapply(with(pars$analyte_pars$number_pars, low:high),
                       function(n) with(pars$agent_pars$ratio_pars, {
                         num = eround(n * low, "low"):eround(n * high,
"high")
                         # data.frame(num = num, abund = gauss(n * mu, 1, n *
sig, num))
                         data.frame(num = num, abund = dgamma(num+1,
(n*mode+1)/entropy+1, 1/entropy))
                       }))

  total_mass <- mapply(outer,
                       lapply(analyte_dstr, function(n) n$mass),
                       lapply(agent_dstr, function(n) n$num *
pars$agent_pars$mass),
                       MoreArgs = list(FUN = "+"),
                       SIMPLIFY = FALSE)

  total_charge <- mapply(outer,
                         lapply(analyte_dstr, function(n) n$mass * 0),
                         lapply(agent_dstr, function(n) n$num *
pars$agent_pars$charge),
                         MoreArgs = list(FUN = "+"),
                         SIMPLIFY = FALSE)

  total_names <- with(pars$analyte_pars$number_pars, low:high)
  total_dimnames <- lapply(1:length(total_mass), function(i) list(mass =
analyte_dstr[[i]]$mass, charge = agent_dstr[[i]]$num *
pars$agent_pars$charge))

  # total_mz <- mapply("/", total_mass, total_charge, SIMPLIFY = FALSE)

  total_abund <- mapply(outer,
                        lapply(analyte_dstr, function(n) n$abund),
                        lapply(agent_dstr, function(n) n$abund),
                        SIMPLIFY = FALSE)

  total_width <- mapply(outer,
                        lapply(analyte_dstr, function(n) n$width),
                        lapply(agent_dstr, function(n) n$num * 0),
                        MoreArgs = list(FUN = "+"),
                        SIMPLIFY = FALSE)
```

```
  names(total_mass) <- total_names
  for(i in 1:length(total_mass)) dimnames(total_mass[[i]]) <-
total_dimnames[[i]]
  names(total_charge) <- total_names
  for(i in 1:length(total_charge)) dimnames(total_charge[[i]]) <-
total_dimnames[[i]]
  # names(total_mz) <- total_names
  # for(i in 1:length(total_mz)) dimnames(total_mz[[i]]) <-
total_dimnames[[i]]
  names(total_abund) <- total_names
  for(i in 1:length(total_abund)) dimnames(total_abund[[i]]) <-
total_dimnames[[i]]
  names(total_width) <- total_names
  for(i in 1:length(total_width)) dimnames(total_width[[i]]) <-
total_dimnames[[i]]

  list(m = total_mass, z = total_charge, h = total_abund, w = total_width)

}
```

"DataCalc.R"

```
# Function for safely generating sequences.
# Native seq function doesn't check for inifinite values.
fseq <- function(from, to, by, length.out){
  if(is.finite(from) & is.finite(to)){
    x <- round(seq(from, to, by) / by) * by
    x <- c(x, rep(NA, length.out - length(x)))
  }
  else{
    x <- rep(NA, length.out)
  }
  x
}

# Function for calculating plot data from parameter data structure.
# Returns list containing data for:
#   total spectrum
#   individual gaussian peaks
#   individual peak m/z and height values
plotData <- function(peak, step, res = 0){

  peak <- peak %>% filter(abund >= max(abund) / 1E6)

  sig <- sapply(abs(peak$width), max, res)
  x_min <- peak$mz - 4 * sig
  x_max <- peak$mz + 4 * sig
  x_length <- round(max((x_max - x_min) / step, na.rm = TRUE)) + 1
  num_points <- length(x_min) * x_length
  max_num_points <- 100000
  if(num_points > max_num_points){
    x_length <- round(max_num_points / length(x_min))
    step <- max(x_max - x_min, na.rm = TRUE) / (x_length - 1)
    print(paste("Step size too small. Changed to:", step))
```

```
  }
  if(!is.finite(x_length)) return()
  x <- mapply(fseq, x_min, x_max, MoreArgs = list(step, x_length), SIMPLIFY =
FALSE)
  y <- mapply(gauss, peak$mz, peak$abund, sig, x)
  x <- as.data.frame(simplify2array(x))

  # Individual peaks categorized by different parameters
  temp <- peak %>% select(-c("mz", "abund", "width"))
  x_data <- melt(cbind(temp, t(x)), id.vars = names(temp), variable.name =
"i", value.name = "x")
  y_data <- melt(cbind(temp, t(y)), id.vars = names(temp), variable.name =
"i", value.name = "y")
  indiv <- full_join(x_data, y_data, by = names(dimnames(x))) %>% filter(y >
0)

  # Total spectrum from adding individual peaks together.
  total <- indiv %>% group_by(x) %>% dplyr::summarize(y = sum(y))

  # Normalize all intensities to max of total spectrum.
  total_area <- sum(total$y) * mean(diff(total$x))
  indiv$y <- indiv$y / max(total$y) * 100
  # peak$y <- peak$y / max(total$y) * 100
  total$y <- total$y / max(total$y) * 100

  list(total = total, indiv = indiv, peak = peak)

}
```

“Plot.R”

```
# Creates interactive plots
iPlotly <- function(p_data, p_var, f_data){

  # Set up generic plot object
  out_plot <- plot_ly(type = "scatter", mode = "lines",
                      colors = colorRamp(c(rgb(1,0,0.5), "red", "orange",
rgb(0.5,1,0), "green",
                                           "turquoise", "cyan", rgb(0,0.5,1),
"blue", "violet", "magenta")))

  # Used to generate x axis tick marks
  xrange <- numeric()
  yrange <- numeric()

  # Add comparison data if it exists
  if(is.data.frame(f_data)){
    out_plot <- out_plot %>%
      add_trace(data = f_data, x = ~x, y = ~y, color = factor("data"), line =
list(color = "black"))
    xrange <- range(c(xrange, f_data$x), na.rm = TRUE)
    yrange <- range(c(yrange, f_data$y), na.rm = TRUE)
  }

  # Add model data if it exists
```

```
  if(length(p_data$total) > 0 & p_var != ""){
    out_plot <- out_plot %>%
      add_trace(data = p_data$total, x = ~(x * sign(x)), y = ~(y * sign(x)),
color = factor("total"), line = list(color = "black")) %>%
      add_trace(data = p_data$indiv %>% group_by_at(vars(-i, -x, -y), .drop =
TRUE),
                x = ~(x * sign(x)), y = ~(y * sign(x)), split =
~factor(get(p_var)), color = ~factor(get(p_var)))
    xrange <- range(c(xrange, p_data$total$x * sign(p_data$total$x)), na.rm =
TRUE)
    yrange <- range(c(yrange, p_data$total$y * sign(p_data$total$x)), na.rm =
TRUE)
  }

  # Hack to create major and minor ticks. Major ticks are on xaxis, minor on
xaxis2
  out_plot %>%
    add_trace(data.frame(x = xrange, y = yrange), xaxis = "x2") %>%
    layout(xaxis = list(title = "m/z", showline = TRUE, showgrid = FALSE,
                        ticks = "outside", ticklen = 10,
                        nticks = 10, range = xrange,
                        tickfont = list(size = 20), titlefont = list(size =
20)),
           xaxis2 = list(overlaying = "x", showgrid = FALSE, showticklabels =
FALSE,
                         ticks = "outside", nticks = 100, range = xrange),
           yaxis = list(title = "Relative Abundance", showline = TRUE,
showgrid = FALSE,
                        ticks = "outside", range = yrange,
                        tickfont = list(size = 20), titlefont = list(size =
20)))

}
```

“model.R”

```
# Default parameters for a component
model <- list(

  analyte_pars = list(

    base_pars = list(
      mass = c(0),
      abund = c(1),
      width = c(0)
    ),

    unit_pars = list(
      mass = c(861800),
      abund = c(1),
      width = c(2000)
    ),

    number_pars = list(
      low = 1,
```

```
      high = 1,
      mode = 1,
      entropy = 0.01
    )

  ),

  agent_pars = list(

    mass = 1,
    charge = 1,
    ratio_pars = list(
      low = 44,
      high = 49,
      mode = 47,
      entropy = 0.1
    )

  )

)
```

VITA

EDUCATION

Doctor of Philosophy, Analytical Chemistry
Purdue University, West Lafayette, IN
August 2016 – Present

Bachelor of Science, Chemistry
Brigham Young University, Provo, UT
August 2010 – April 2011, January 2014 – April 2016

RESEARCH

Development of ion trap technology for mass spectrometry of biomolecular complexes
Purdue University, West Lafayette, IN
Adviser: Dr. Scott McLuckey
August 2016 – December 2020

Fabrication of nanoscale devices using DNA origami
Brigham Young University, Provo, UT
Adviser: Dr. Adam Woolley
August 2010 – April 2011, January 2014 – April 2016

TEACHING

Teaching Assistant in Department of Chemistry
Purdue University, West Lafayette, IN
Fall 2016 CHM 11100: General Chemistry 1
Spring 2017 CHM 37401: Physical Chemistry Laboratory
Fall 2017 CHM 11500: General Chemistry 1 (for Engineering students)

Teaching Assistant in Department of Chemistry
Brigham Young University, Provo, UT
Fall 2014 Chem 111: Principles of Chemistry 1
Winter 2015 Chem 101 & 105: Exploratory Lab Section
Winter 2016 Chem 112: Principles of Chemistry 2

PRESENTATIONS & POSTERS

Improving mass measurements of protein complexes through IR activation coupled with charge reduction ion/ion reactions. Kenneth W. Lee; Christopher P. Harrilal; Liangxuan Fu; Gregory S. Eakins; Scott A. McLuckey. TOH pm, June 2, Proceedings of the 68th ASMS Conference on Mass Spectrometry and Allied Topics, Online Reboot, June 1–12, 2020.

Increasing the Mass Range of Ion-Ion Reactions in a Quadrupole Ion Trap with Waveform Switching. Kenneth W. Lee; Gregory S. Eakins; Mark S. Carlsen; Scott A. Mcluckey, MP 487, June 2, Proceedings of the 67th ASMS Conference on Mass Spectrometry and Allied Topics, Atlanta, GA, June 2–6, 2019.

AWARDS & HONORS

W. Brooks Fortune Fellowship
Purdue University, West Lafayette, IN
Fall 2017

Ross Fellowship
Purdue University, West Lafayette, IN
Fall 2016 – Spring 2017

Dean's List
Brigham Young University, Provo, UT
Winter 2014

Undergraduate Research Award
Brigham Young University, Provo, UT
Winter 2014, Spring/Summer 2014, Fall 2014, Winter 2015, Spring/Summer 2015, Fall 2015, Winter 2016

SKILLS

Programming and circuitry
Education of C++, including object-oriented programming and data structures (Brigham Young University Fall/Winter 2014)
Experience with R for data analysis and calculations/models of chemical processes
Experience with Python for designing desktop programs
Experience with SIMION and Lua programming language
Experience with Autodesk Inventor
Experience programming microcontrollers
Experience designing and programming an analytical instrument controller

PUBLICATIONS

Lee, Kenneth W.; Harrilal, Christopher P.; Fu, Liangxuan; Eakins, Gregory S.; McLuckey, Scott A. Digital ion trap mass analysis of high mass protein complexes using IR activation coupled with ion/ion reactions. *International Journal of Mass Spectrometry*. **2020**, 458, 116437.

Lee, Kenneth W.; Eakins, Gregory S.; Carlsen, Mark S.; McLuckey, Scott A. Ion trap operational modes for ion/ion reactions yielding high mass-to-charge product ions. *International Journal of Mass Spectrometry*. **2020**, 451, 116313.

Foreman, David J.; Bhanot, Jay S.; Lee, Kenneth W.; McLuckey, Scott A. Valet Parking for Protein Ion Charge State Concentration: Ion/Molecule Reactions in Linear Ion Traps. *Analytical Chemistry*. **2020**, 92(7), 5419-5425.

Lee, Kenneth W.; Eakins, Gregory S.; Carlsen, Mark S.; McLuckey, Scott A. Increasing the Upper Mass/Charge Limit of a Quadrupole Ion Trap for Ion/Ion Reaction Product Analysis via Waveform Switching. *Journal of the American Society for Mass Spectrometry.* **2019**, 30(6), 1126-1132.

Johnson, Joshua T.; Lee, Kenneth W.,; Bhanot, Jay S.; McLuckey, Scott A. A Miniaturized Fourier Transform Electrostatic Linear Ion Trap Mass Spectrometer: Mass Range and Resolution. *Journal of the American Society for Mass Spectrometry*. **2019**, 30(4), 588-594.

Dziekonski, Eric T.; Johnson, Joshua T.; Lee, Kenneth W.; McLuckey, Scott A. Determination of Collision Cross Sections Using a Fourier Transform Electrostatic Linear Ion Trap Mass Spectrometer. *Journal of the American Society for Mass Spectrometry*. **2018**, 29(2), 242-250.

Dziekonski, Eric T.; Johnson, Joshua T.; Lee, Kenneth W.; McLuckey, Scott A. Fourier-Transform MS and Closed-Path Multireflection Time-of-Flight MS Using an Electrostatic Linear Ion Trap. *Analytical Chemistry*. **2017**, 89(20), 10965-10972.