Supplemental Information

Logical MS/MS Scans: A New Set of Operations for Tandem Mass Spectrometry

Dalton T. Snyder; Lucas J. Szalwinski; J. Mitchell Wells; R. Graham Cooks*

*Corresponding author: cooks@purdue.edu

Table of Contents

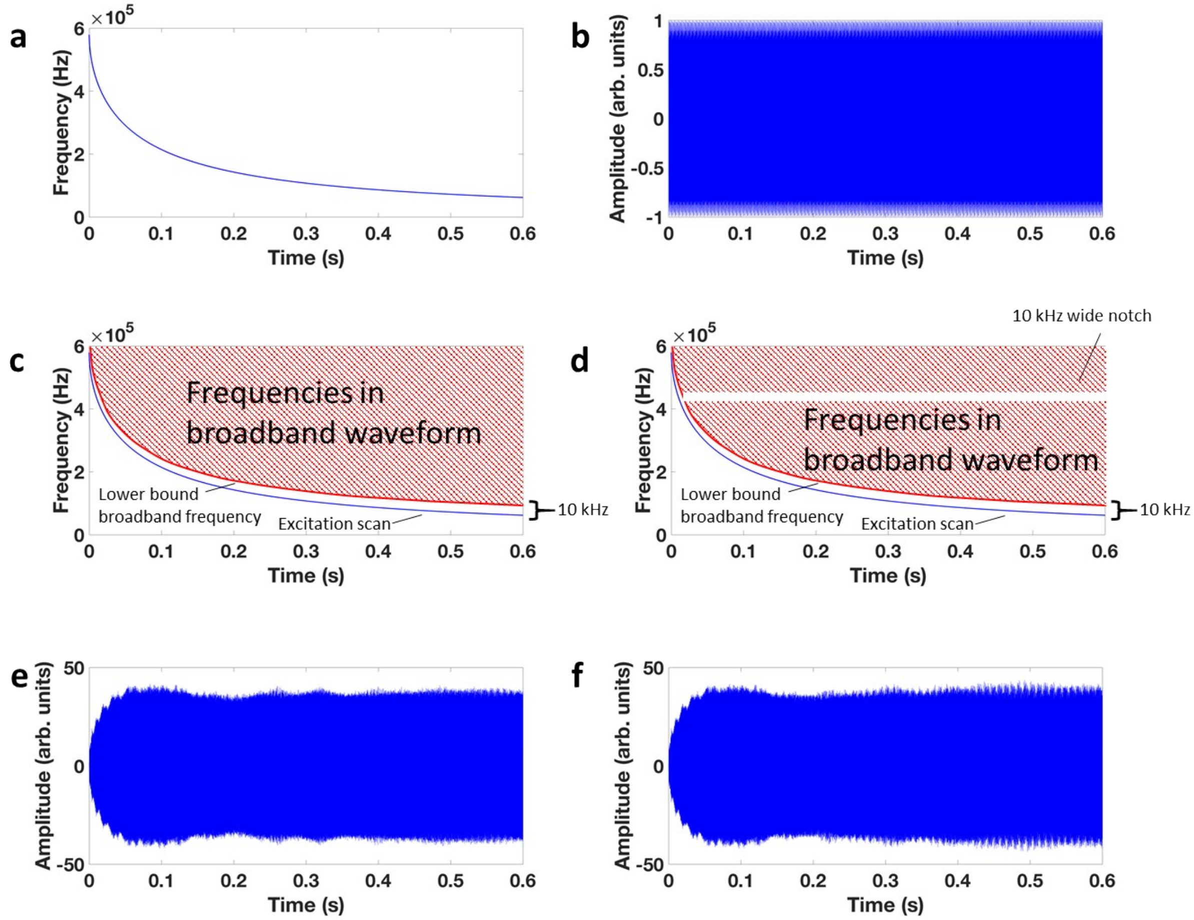

Figure S1: Illustrative waveforms used in this study: (a) frequency vs. time profile for an inverse Mathieu q scan over 600 ms from q = 0.908 to q = 0.15, (b) a broadband sum of sines waveform with 1 kHz frequency spacing from 300 kHz (q = 0.654) to 50 kHz (q = 0.12) and a quadratic phase relationship with respect to frequency, (c) frequency vs. time relationship for a broadband waveform with frequency lower bound 10 kHz higher than the inverse Mathieu q scan in (a), (d) the same frequency profile as (c) but with a 10 kHz wide notch, (e) a waveform with the profile in (c), and (f) a waveform with the profile in (d) where the notch at 157 kHz is 10 kHz wide (used for a NOT precursor scan). Voltage is shown in arbitrary units because they end up scaled by the waveform generator.

```
% Program: Inverse_Mathieu_q_Scan.m
% Calculates an Inverse Mathieu q Scan, i.e. an ac frequency sweep with
% approximately linear mass scale

% Define variables
scan_time = 0.6;              % mass scan time in seconds
begin_q = 0.908;              % starting Mathieu q value
end_q = 0.15;                 % ending Mathieu q value
sampling_rate = 5000000;      % sampling rate of waveform - must match
                              % function generator (Sa/s)
rf_frequency = 1166000;       % trap rf frequency in Hz
num_points = ceil(sampling_rate * scan_time);   % number of points in the
                                                % waveform
time = linspace(0, num_points-1, num_points)*scan_time/num_points;
                              % time variable
begin_amplitude = 1;          % p-p voltage to start at if doing an
                                          % amplitude ramp
end_amplitude = 1;            % p-p voltage to end at if doing an amplitude
                                          % ramp
phi(1) = 0;                   % initial phase of waveform, best to set at 0
                                          % so scan starts at 0 voltage

% Calculate Mathieu q values as a function of time
% Assume sweep according to q = k / (t-j)
% where q is Mathieu q value to interrogate,
% t is time, and k and j are constants to be calculated
j = end_q*scan_time / (end_q - begin_q);
k = -begin_q*j;
q_values = k ./ (time - j);

% Assume linear ramp of ac amplitude
% If begin and end amplitude are the same, then amplitude is constant
amplitudes = linspace(begin_amplitude,end_amplitude,num_points);

% Preallocate memory for frequency, beta, and waveform voltage as a function
% of time
frequencies = zeros(num_points,1);
betas = zeros(num_points,1);
waveform = zeros(num_points,1);

% Use a phase accumulator (phi) to do a nonlinear frequency sweep, convert
% Mathieu q to beta, then to frequency, and finally to phase accumulator
for i = 1:num_points
    betas(i) = beta_calculator(q_values(i));
    frequencies(i) = betas(i)*rf_frequency/2;
    waveform(i) = amplitudes(i)*sin(phi(i));
    delta = 2*pi*frequencies(i)/sampling_rate;
    phi(i+1) = phi(i) + delta;
end
```

Program S1: Program for calculating an inverse Mathieu q scan with (in this case) 600 ms mass scan time and start and end Mathieu q values of 0.908 and 0.15, respectively.

```
function [beta] = beta_calculator(q)

% Function: beta_calculator
% Accepts an ion's Mathieu q value and calculates the ion's beta value

% Initial guess
beta = 0.5;
prev_beta = 0;

% Bounds
left_bound = 0;
right_bound = 1;

% Tolerance defines accuracy of result
tolerance = 0.00001;
LHS_minus_RHS = 1;      % LHS = left hand side of the equation for calculating
                        % beta; RHS = right hand side

% Iterate the calculation until the difference between the LHS and RHS of the
equation for calculating beta is below a specified tolerance
while (abs(beta - prev_beta) > tolerance)
    % Left hand side of beta equation
    LHS = beta^2;
    q_sq = q^2;

    % Right hand side of beta equation
    RHS = q_sq/((beta+2)^2 - q_sq/((beta+4)^2 - q_sq/((beta+6)^2 -
q_sq/((beta+8)^2-q_sq/(beta+10)^2)))) + q_sq/((beta-2)^2 - q_sq/((beta-4)^2 -
q_sq/((beta-6)^2-q_sq/((beta-8)^2-q_sq/(beta-10)^2))));

    LHS_minus_RHS = LHS - RHS;

    % Guess not close enough
    if LHS_minus_RHS < 0
        prev_beta = beta;
        beta = (beta + right_bound) / 2;
        left_bound = prev_beta;
    elseif LHS_minus_RHS > 0
        prev_beta = beta;
        beta = (beta + left_bound) / 2;
        right_bound = prev_beta;
    else
    % do nothing, guess was close enough
    end
end
end
```

Program S2: Program for converting from Mathieu q to parameter beta (which is directly proportional to secular frequency).

```
% Program: sum_sine_excitation
% Creates a broadband waveform with no notches for excitation
% or ejection of ions over a wide mass range

sampling_rate = 5000000;                    % waveform generator sampling rate
                                            % (Sa/s)
rf_freq = 1166000;                    % trapping rf frequency in Hz
start_freq = 300000;                        % where the notch starts (Hz)
end_freq = 50000;                     % where the notch ends (Hz)
freq_resolution = 1000;                     % difference between adjacent
                                            % frequencies (Hz)
waveform_length_s = 0.1;                    % length of waveform in s
phase_fudge_factor = 0.001;           % determines how phases are distributed
num_points = round(waveform_length_s*sampling_rate);  % number of points in
                                                        % the waveform

% Initialize arrays for the waveform, frequency vs. time, and phase as a
% function of frequency
master_waveform = zeros(num_points,1);
frequencies = linspace(start_freq,end_freq,num_frequencies);
num_frequencies = length(frequencies);
phases = zeros(num_frequencies,1);

% Distribute phases so that master waveform has flat amplitude profile
% Assumes phases are quadratically related to frequency
for i=1:num_frequencies
    phases(i) = (frequencies(i)-
frequencies(1))^2*waveform_length_s/(2*(frequencies(num_frequencies)-
frequencies(1))*phase_fudge_factor);
end

% Make time array
time = linspace(0,waveform_length_s,waveform_length_s*sampling_rate).';

% Build master waveform
for i=1:num_frequencies
    master_waveform = master_waveform +
      sin(2*pi*frequencies(i).*time+phases(i));
end

% normalize amplitude to 1
master_waveform = (1/max(master_waveform))*master_waveform;
```

Program S3: Program for creating a broadband sum of sines waveform for ion excitation. Phases are distributed quadratically vs. frequency to ensure a flat amplitude profile.

```
% Program: Not_Scan (precursor scan version)
% Calculates a notched broadband waveform that varies with time so that the
% lowest frequency at any given time point is above the corresponding
% frequency for the specified inverse Mathieu q scan. This prevents
% precursor ions from being ejected by the broadband waveform before they are
% fragmented by the inverse Mathieu q scan. The notch is fixed and placed so
% as to prevent the ejection of a selected product ion, thereby ejecting all
% product ions except the selected ion.

% Define variables
scan_time = .6;                          % scan time in seconds
begin_q = 0.908;                   % Mathieu q value to start at
end_q = 0.15;                            % Mathieu q value to end at
sampling_rate = 5000000;           % sampling rate of waveform (Sa/s)
rf_frequency = 1166000;            % trap rf frequency in Hz
num_points = ceil(sampling_rate * scan_time);  % number of points in waveform
time = linspace(0, num_points-1, num_points)*scan_time/num_points;
                              % time variable
frequency_resolution = 1000;       % spacing between adjacent frequencies
(Hz)
distance_from_lower_bound = 10000;       % frequency distance between the
inverse
                              % Mathieu q scan and the lowest frequency
                              % in the broadband waveform
phase_fudge_factor = 0.0001;       % used for phase overmodulation
notch_frequency = 157000;                % center frequency of notch in Hz
notch_width = 10000;                           % frequency width of notch in
Hz

% Calculate Mathieu q values as a function of time for an inverse Mathieu q
% scan
% Assume sweep according to q = k / (t-j)
j = end_q*scan_time / (end_q - begin_q);  k = -begin_q*j;
q_values = k ./ (time - j);

% Convert from Mathieu q to frequency; 'lower_bound_frequencies' contains the
% lower bound frequency of the broadband waveform as a function of time, i.e.
% the frequency that the inverse Mathieu q scan is applying as a function of
% time
lower_bound_frequencies = zeros(num_points,1);
betas = zeros(num_points,1);
for i = 1:num_points
    betas(i) = beta_calculator(q_values(i));
    lower_bound_frequencies(i) = betas(i)*rf_frequency/2;
end

% Build frequencies array
num_frequencies = floor(abs(rf_frequency/2-
lower_bound_frequencies(end))/frequency_resolution); % total number of
                                                     % frequencies in waveform
```

```
frequencies =
linspace(rf_frequency/2,lower_bound_frequencies(end),num_frequencies);
                  %

% Distribute phases so that master waveform has flat amplitude profile
phases = zeros(num_frequencies,1);
for i=1:num_frequencies
    phases(i) = (frequencies(i)-
frequencies(1))^2*scan_time/(2*(frequencies(num_frequencies)-
frequencies(1))*phase_fudge_factor);
end

% Build final waveform point by point
waveform = zeros(num_points,1);
for i=1:num_points
    for n=1:length(frequencies)
        % This frequency is above the lower bound and is not in the notch, so
        % include it!
        if ((frequencies(n) > lower_bound_frequencies(i) +
            distance_from_lower_bound) && ~((frequencies(n) < notch_frequency
            + notch_width/2) && (frequencies(n) > notch_frequency -
            notch_width/2)))
            waveform(i) =
            waveform(i) + sin(2*pi*frequencies(n)*time(i) + phases(n));
        end
    end
end
```

Program S4: Program for calculating a notched broadband waveform for a NOT scan (precursor scan variant). The broadband waveform varies as a function of time so that the included frequencies are always above the corresponding inverse mathieu q scan frequency at each time point. A fixed notch is also implemented so as to prevent the ejection of a selected product ion. Thus, this scan (in conjunction with an inverse Mathieu q scan for precursor ion excitation) detects all precursor ions that do not exclusively produce the selected product ion. To create a NOR scan broadband waveform, two notches are implemented instead of one. For neutral loss variants, no notches are used and instead the neutral loss products are neutralized on the y rods by an inverse Mathieu q scan.