Overview

The m.files add\_Ca\_file.m and calculate\_Ca\_ratio.m take the light intensity and the wavelength arrays from a set of measurements and determines the Calcium ratio from it.

Additional resources

Need more help?

Check the resources, and then see Ken

Main content

**Part 1: How to use add\_Ca\_records**

1. Adjust the location of the measurements and the names of the measurement result files.

dir\_string='z:\c. chung\trabrec\20140712b';  
file\_strings={ ...  
    'test10.slc', ...  
    'test11.slc', ...  
    'test12.slc'};

1. The data is read in using the function *load\_slcontrol\_file(...)* and placed in structure d. Function *transform\_slcontrol\_record(d)* changes the voltage values into the real SI units values. Finally this data is used by the *calculate\_ca\_ratio* function to determine the Calcium ratio and place this data into the structure.  
   If you want to see some of the results, the code automatically plots the figures of the force, Calcium ratio and the stimulus. More figures can be added to this, just put in another subplot and adjust the no\_of\_panels.

% Parameters to do with figure  
color\_map=jet(numel(file\_strings));  
no\_of\_panels\_high=4;  
  
figure(1);  
clf;  
  
for i=1:numel(file\_strings)  
    d=load\_slcontrol\_file(fullfile(dir\_string,file\_strings{i}));  
    d=transform\_slcontrol\_record(d,0);  
    Ca\_struct=calculate\_Ca\_ratio(d);  
    d.Ca\_ratio=Ca\_struct.Ca\_ratio;  
      
    subplot(no\_of\_panels\_high,1,1);  
    hold on;  
    plot(d.time,d.force,'-', ...  
        'Color',color\_map(i,:));  
  
    subplot(no\_of\_panels\_high,1,2);  
    hold on;  
    plot(d.time,d.Ca\_ratio,'-', ...  
        'Color',color\_map(i,:));  
  
    subplot(no\_of\_panels\_high,1,3);  
    hold on;  
    plot(d.time,d.stimulus,'-', ...  
        'Color',color\_map(i,:));  
end

**Part 2: How to calculate\_CA\_ratio\_works**

1. **function definition**  
   The function is defined and has the option to receive a variable number of inputs, varargin (variable argument input). this gives the function some more flexibility.

function calcium\_structure = calculate\_Ca\_ratio(data\_structure,varargin)

1. **Extra variables**  
   Some additional variables are defined that can be adapted by putting these in the place of varargin when using the function.  
   If these additional variables have received new values in the function, parse\_pv\_pairs places them into the parameters.

params.good\_pmt\_value\_start=5; % Data point after sq wave transition to start median calc  
params.threshold\_wavelength\_command=0.1; % Values lower than the threshold are points where the system is still setting and are removed   
params.test\_for\_plateauing = 0; % Make this larger than 0 to turn on the test for plateauing, this helps determine good\_pmt\_value\_start  
  
% Replace predefined values with values defined in the function  
params = parse\_pv\_pairs(params,varargin);

1. **Find the beginning number of junk points**  
   The first few values in each measurement are taken when the system is still being set up, because of this the first few values are faulty and should be removed. This part identifies the amount of values that are bad.

% Find the number of beginning points for which pmt values are not actual measurements  
begin\_junk = find(data\_structure.wavelength\_command<params.threshold\_wavelength\_command);  
begin\_junk\_pts = length(begin\_junk);

1. **Remove the junk points from the measurement**  
   The first few numbers, equal to the amount of junk points, are removed from the measurement data.

% Removing the first "begin\_junk\_pts" data points since the monochromator   
% is still setting during this time  
pmt\_clean    = data\_structure.light\_intensity(begin\_junk\_pts+1:end);  
sqwave\_clean = data\_structure.wavelength\_command(begin\_junk\_pts+1:end);

1. **Half wavelength**  
   Since the wavelength can be varied by the user if so required, the wavelength is determined by the function to use later on. This is done by checking where the wavelength is low for the first time and where it is high for the first time and than determining the difference between these.

% Find the number of sample points in half a wave  
samp\_lower  = find(sqwave\_clean(:,1)<mean(sqwave\_clean(:,1)),1,'first');  
samp\_higher = find(sqwave\_clean(:,1)>mean(sqwave\_clean(:,1)),1,'first');  
half\_wave\_duration = abs(samp\_lower-samp\_higher);

1. **Create restore length pieces**  
   Since the first few junk points have been removed, the length of the measurement is now shorter than it used to be. To restore this, the values of exactly one full wavelength back are copied.

% Since the last two values are missing for the last block, copy the last  
% two values of the second last block  
restore\_length = pmt\_clean(end-(2\*half\_wave\_duration):end-(2\*half\_wave\_duration-begin\_junk\_pts+1));  
restore\_sqwave = sqwave\_clean(end-(2\*half\_wave\_duration):end-(2\*half\_wave\_duration-begin\_junk\_pts+1));

1. **Restore length**  
   Actually restore the measurement length by patching the wavelength and pmt arrays.

% Use the previously copied piece to patch the last block  
pmt\_restored    = [pmt\_clean; restore\_length];  
sqwave\_restored = [sqwave\_clean; restore\_sqwave];

1. **Calculate median values**  
   Calculates the median values for each half wavelength. This is done by first reshaping the light intensity data into colums of alternating high and low wavelengths. For instance if one half wavelength is 25 values long, the reshaped matrix will contain vectors that are 25 values long and the matrix itself is length(pmt)/25 wide.  
   After that the median value of each vector is determined.

% Calculate the medians of the valid data points (i.e. data points where  
% wavelength is not switching)  
pmt\_reshaped = reshape(pmt\_restored, half\_wave\_duration, (length(pmt\_restored)/half\_wave\_duration));  
pmt\_medians  = median(pmt\_reshaped(params.good\_pmt\_value\_start:half\_wave\_duration,:));

1. **Find high and low parts of the squarewave**  
   In order to split the light intensity data into two arrays, one where the wavelength was low and the other where it was high, we must first find where the squarewave where low and high.  
   First the vectors where the results will be placed in are created. This is done to save processing speed in MATLAB.  
   Now for each value in the squarewave where the wavelength was lower than the mean value of the wavelengths, this value becomes 1 for the sqwave\_low and a NaN for sqwave\_high. Vice versa happens for the high values.

% Find the points for which the squarewave is low and where it is high  
sq\_mean = mean(sqwave\_restored);  
sqwave\_low = zeros(length(sqwave\_restored),1);  
sqwave\_high = zeros(length(sqwave\_restored),1);  
  
for i = 1:length(sqwave\_restored)  
    if sqwave\_restored(i) < sq\_mean  
       sqwave\_low(i)  = 1;  
       sqwave\_high(i) = NaN;  
    else sqwave\_low(i)  = NaN;  
         sqwave\_high(i) = 1;  
    end  
end

1. **Reshape high and low squarewaves**  
   Like the light intensity data, both high and low parts of the squarewave are reshaped into matrices with height of one half wavelength.

% reshape the low/high squarewave data to get columns of low/high  
sqwave\_low\_reshape  = reshape(sqwave\_low, half\_wave\_duration, ...  
    (length(sqwave\_low)/half\_wave\_duration));  
sqwave\_high\_reshape = reshape(sqwave\_high, half\_wave\_duration, ...  
    (length(sqwave\_high)/half\_wave\_duration));

1. **Place median wavelengths into the high and low matrices**  
   The respectively low and high wavelength wavelengths are copied into the low and high squarewave matrices. This means that at this point, the low part of the Ca signal will have the median wavelength value of the first vector in each data point in the first vector. The second vector will contain NaN's (since the wavelength was high here). The third will contain the medians of the original third vector, etc.

% Predefine matrices to reduce runtime  
part\_Ca\_low  = zeros(half\_wave\_duration,length(pmt\_medians));  
part\_Ca\_high = zeros(half\_wave\_duration,length(pmt\_medians));  
  
% Place the low/high medians in respectively the high/low squarewave  
% columns  
for i = 1:length(pmt\_medians)  
  part\_Ca\_low(:,i)  = pmt\_medians(i).\*sqwave\_high\_reshape(:,i);  
  part\_Ca\_high(:,i) = pmt\_medians(i).\*sqwave\_low\_reshape(:,i);  
end

1. **Duplicate and interpolate columns**  
   Both low and high parts of the Ca signal now still alternate between containing values and containing NaN's. This part of the code copies respectively the previous and the next data vector to the next and previous vector.  
   This would look something like this:  
     
   Part\_Ca\_low:  
   NaN          a          NaN          b          NaN          c  
   NaN          a          NaN          b          NaN          c  
   NaN          a          NaN          b          NaN          c  
   NaN          a          NaN          b          NaN          c  
   ...  
     
   Part\_Ca\_low (after copying):  
   a              a            b             b            c            c  
   a              a            b             b            c            c  
   a              a            b             b            c            c  
   a              a            b             b            c            c  
   ...  
   For the low part, each even vector is copied and placed in the previous odd vector.  
   For the high part, each odd vector is copied and placed in the next even vector.

% Duplicates and interpolates the columns  
for i=1:(length(pmt\_medians)/2)  
  part\_Ca\_low(:,2\*i-1) = part\_Ca\_low(:,2\*i);  
  part\_Ca\_high(:,2\*i)  = part\_Ca\_high(:,2\*i-1);  
end

1. **Reshape the low and high Calcium signals back into arrays**

% Reshape the Calcium signals back to vectors  
Ca\_340 = reshape(part\_Ca\_low, length(pmt\_restored), 1);  
Ca\_380 = reshape(part\_Ca\_high, length(pmt\_restored), 1);

1. **Calculate the actual Calcium ratio**  
   Now that both low and high Calcium signals are known, the ratio of low/high can be determined.

% Calculate the ratio of low over high Calcium signals  
Ca\_ratio = Ca\_340./Ca\_380;  
  
calcium\_structure.Ca\_340   = Ca\_340;  
calcium\_structure.Ca\_380   = Ca\_380;  
calcium\_structure.Ca\_ratio = Ca\_ratio;

**Part 3: Optional Test for Plateauing**

The final part of the code will not be run most of the time, but it is possible to use it to check for plateauing and see any delay in the wavelength signals.  
  
The code takes each low part of the wavelength and determines the average value for each point of half of a wavelength. Likewise happens for the high part of the wavelength.  
  
The standard deviations of both high and low parts are determined as well.  
  
Finally a figure is plotted containing the averaged data, average minus std and average plus std.

%% Test for plateauing

if params.test\_for\_plateauing > 0

pmt\_rising = sqwave\_low.\*pmt\_restored;

pmt\_descending = sqwave\_high.\*pmt\_restored;

pmt\_rise\_clean = pmt\_rising(~isnan(pmt\_rising));

pmt\_desc\_clean = pmt\_descending(~isnan(pmt\_descending));

pmt\_rise\_reshape = reshape(pmt\_rise\_clean, half\_wave\_duration, (length(pmt\_rise\_clean)/half\_wave\_duration));

pmt\_desc\_reshape = reshape(pmt\_desc\_clean, half\_wave\_duration, (length(pmt\_desc\_clean)/half\_wave\_duration));

mean\_rise = mean(pmt\_rise\_reshape,2);

mean\_desc = mean(pmt\_desc\_reshape,2);

std\_rise = std(pmt\_rise\_reshape,0,2);

std\_desc = std(pmt\_desc\_reshape,0,2);

plateau = [mean\_rise; mean\_desc];

STD = [std\_rise; std\_desc];

part\_sqwave = sqwave\_restored(1:2\*half\_wave\_duration);

points = 1:2\*half\_wave\_duration;

    figure

    plot(points, plateau, 'b', points, (part\_sqwave-1)/(max(part\_sqwave)), 'r', points, plateau-STD, 'm', points, plateau+STD, 'm')

end

**Part 4: Complet MATLAB file of function calculate\_Ca\_ratio**

function calcium\_structure = calculate\_Ca\_ratio(data\_structure,varargin)

params.good\_pmt\_value\_start=5; % Data point after sq wave transition to start median calc

params.threshold\_wavelength\_command=0.1; % Values lower than the threshold are points where the system is still setting and are removed

params.test\_for\_plateauing = 0; % Make this larger than 0 to turn on the test for plateauing, this helps determine good\_pmt\_value\_start

% Replace predefined values with values defined in the function

params = parse\_pv\_pairs(params,varargin);

%% Finds the low and high Calcium signals and calculates the ratio of low/high

% Koen van de Poll

% 6/16/2014

% last update on 7/17/2014 by Koen

% Campbell Muscle Lab

%% Determine low/high Calcium ratio

%%% GET DATA

% Find the number of beginning points for which pmt values are not actual measurements

begin\_junk = find(data\_structure.wavelength\_command<params.threshold\_wavelength\_command);

begin\_junk\_pts = length(begin\_junk);

%%% CORRECT FOR LOST POINTS AS THE SYSTEM TURNS ON

% Removing the first "begin\_junk\_pts" data points since the monochromator

% is still setting during this time

pmt\_clean    = data\_structure.light\_intensity(begin\_junk\_pts+1:end);

sqwave\_clean = data\_structure.wavelength\_command(begin\_junk\_pts+1:end);

% Find the number of sample points in half a wave

samp\_lower  = find(sqwave\_clean(:,1)<mean(sqwave\_clean(:,1)),1,'first');

samp\_higher = find(sqwave\_clean(:,1)>mean(sqwave\_clean(:,1)),1,'first');

half\_wave\_duration = abs(samp\_lower-samp\_higher);

% Since the last two values are missing for the last block, copy the last

% two values of the second last block

restore\_length = pmt\_clean(end-(2\*half\_wave\_duration):end-(2\*half\_wave\_duration-begin\_junk\_pts+1));

restore\_sqwave = sqwave\_clean(end-(2\*half\_wave\_duration):end-(2\*half\_wave\_duration-begin\_junk\_pts+1));

% Use the previously copied piece to patch the last block

pmt\_restored    = [pmt\_clean; restore\_length];

sqwave\_restored = [sqwave\_clean; restore\_sqwave];

%%% CORRECT FOR STEP RESPONSE

% Calculate the medians of the valid data points (i.e. data points where

% wavelength is not switching)

pmt\_reshaped = reshape(pmt\_restored, half\_wave\_duration, (length(pmt\_restored)/half\_wave\_duration));

pmt\_medians  = median(pmt\_reshaped(params.good\_pmt\_value\_start:half\_wave\_duration,:));

% Find the points for which the squarewave is low and where it is high

sq\_mean = mean(sqwave\_restored);

sqwave\_low = zeros(length(sqwave\_restored),1);

sqwave\_high = zeros(length(sqwave\_restored),1);

for i = 1:length(sqwave\_restored)

    if sqwave\_restored(i) < sq\_mean

       sqwave\_low(i)  = 1;

       sqwave\_high(i) = NaN;

    else sqwave\_low(i)  = NaN;

         sqwave\_high(i) = 1;

    end

end

% reshape the low/high squarewave data to get columns of low/high

sqwave\_low\_reshape  = reshape(sqwave\_low, half\_wave\_duration, ...

    (length(sqwave\_low)/half\_wave\_duration));

sqwave\_high\_reshape = reshape(sqwave\_high, half\_wave\_duration, ...

    (length(sqwave\_high)/half\_wave\_duration));

% Predefine matrices to reduce runtime

part\_Ca\_low  = zeros(half\_wave\_duration,length(pmt\_medians));

part\_Ca\_high = zeros(half\_wave\_duration,length(pmt\_medians));

% Place the low/high medians in respectively the high/low squarewave

% columns

for i = 1:length(pmt\_medians)

  part\_Ca\_low(:,i)  = pmt\_medians(i).\*sqwave\_high\_reshape(:,i);

  part\_Ca\_high(:,i) = pmt\_medians(i).\*sqwave\_low\_reshape(:,i);

end

% Duplicates and interpolates the columns

for i=1:(length(pmt\_medians)/2)

  part\_Ca\_low(:,2\*i-1) = part\_Ca\_low(:,2\*i);

  part\_Ca\_high(:,2\*i)  = part\_Ca\_high(:,2\*i-1);

end

% Reshape the Calcium signals back to vectors

Ca\_340 = reshape(part\_Ca\_low, length(pmt\_restored), 1);

Ca\_380 = reshape(part\_Ca\_high, length(pmt\_restored), 1);

%%% CALCULTE RATIO

% Calculate the ratio of low over high Calcium signals

Ca\_ratio = Ca\_340./Ca\_380;

calcium\_structure.Ca\_340   = Ca\_340;

calcium\_structure.Ca\_380   = Ca\_380;

calcium\_structure.Ca\_ratio = Ca\_ratio;

%% Test for plateauing

if params.test\_for\_plateauing > 0

pmt\_rising = sqwave\_low.\*pmt\_restored;

pmt\_descending = sqwave\_high.\*pmt\_restored;

pmt\_rise\_clean = pmt\_rising(~isnan(pmt\_rising));

pmt\_desc\_clean = pmt\_descending(~isnan(pmt\_descending));

pmt\_rise\_reshape = reshape(pmt\_rise\_clean, half\_wave\_duration, (length(pmt\_rise\_clean)/half\_wave\_duration));

pmt\_desc\_reshape = reshape(pmt\_desc\_clean, half\_wave\_duration, (length(pmt\_desc\_clean)/half\_wave\_duration));

mean\_rise = mean(pmt\_rise\_reshape,2);

mean\_desc = mean(pmt\_desc\_reshape,2);

std\_rise = std(pmt\_rise\_reshape,0,2);

std\_desc = std(pmt\_desc\_reshape,0,2);

plateau = [mean\_rise; mean\_desc];

STD = [std\_rise; std\_desc];

part\_sqwave = sqwave\_restored(1:2\*half\_wave\_duration);

points = 1:2\*half\_wave\_duration;

    figure

    plot(points, plateau, 'b', points, (part\_sqwave-1)/(max(part\_sqwave)), 'r', points, plateau-STD, 'm', points, plateau+STD, 'm')

end