Documentation for NanobubbleDigitizer: A MATLAB based program for detection of bubble induced current blockage signals inside nanopores.

Created by: Soumyadeep Paul

(Confidential: For internal use in Daiguji and Hsu Lab)

Date: 2021/07/15

% NanobubbleDigitizer data analysis package licence details:

%

% Copyright 2009-2014 Soumyadeep Paul (soumyadeep.paul@thml.t.u-tokyo.ac.jp)

%

% This file is part of NanobubbleDigitizer data analysis package.

%

% NanobubbleDigitizer is a MATLAB package designed to detect discrete

% bubble events from transient current traces. The bubble features

% extracted from the current data using this code can be used to analyze

% nanoscopic bubble dynamics confined within solid-state nanopores. This

% package is released under the GNU GPL.

%

% The copyright of NanobubbleDigitizer is held by Soumyadeep Paul.

%

% NanobubbleDigitizer is released under the GNU GPL. NanobubbleDigitizer

% is free software: you can redistribute it and/or modify it under the

% terms of the GNU General Public License (version 3) as published

% by the Free Software Foundation.You should have received a copy of the

% GNU General Public Licence along with arb (see file licence.txt after

% unpacking). If not, see <http://www.gnu.org/licences/>.

%

% NanobubbleDigitizer is distributed in the hope that it will be useful,

% but WITHOUT ANY WARRANTY; without even the implied warranty of

% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General

% Public Licence for more details.

%

% For full details of arb's licence see the licence file in the main

% directory.

1. Overview of package

NanobubbleDigitizer is a nanopore data analysis interface, which can be used to analysis the current-time data of bubble generation in solid-state nanopore.

Signal characteristics are very sensitive to pore size, which requires a tunable algorithm with control over many parameters. For example, with increase in pore size, the range of blockage durations can be between tens of nanoseconds to more than 1200 ns. Whenever, the range of bubble sizes is so big, a more complex algorithm is needed. Also, the error of algorithm is increased as well.

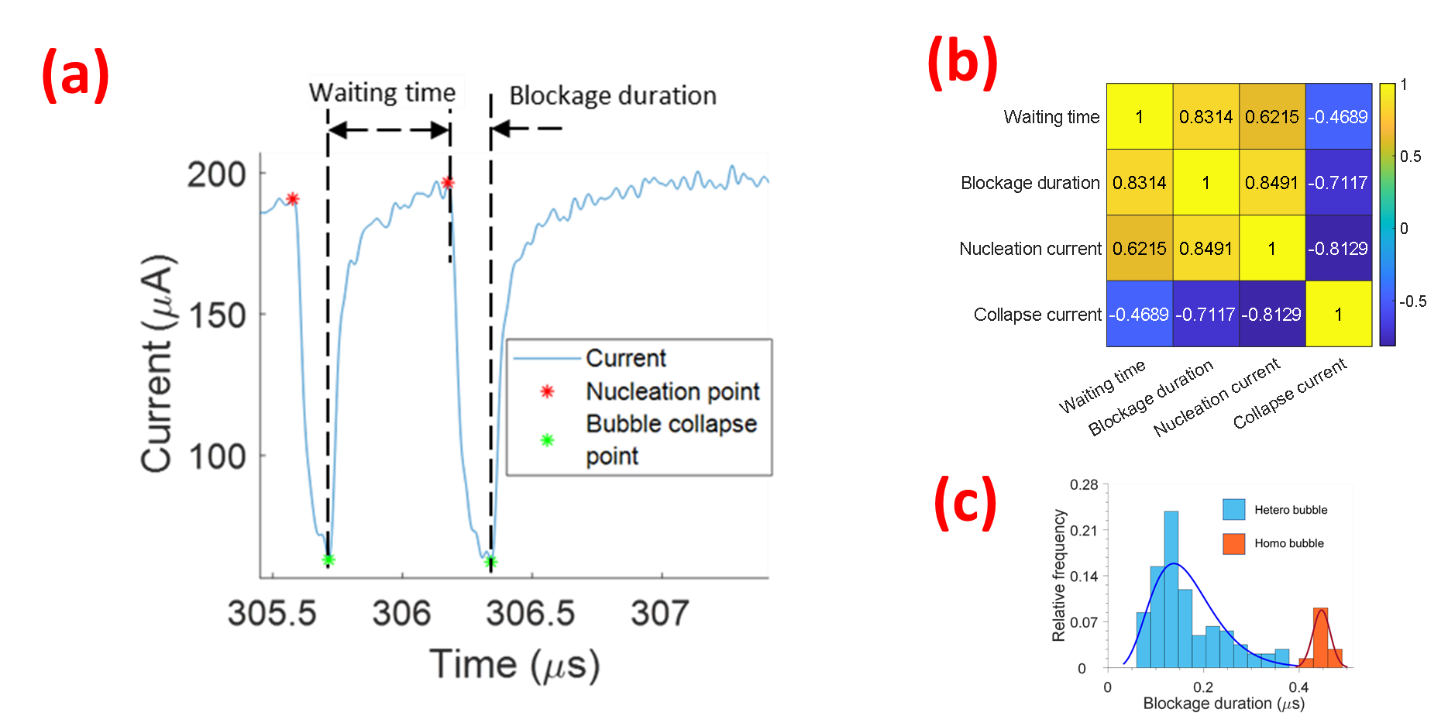


Fig.1 (a) Current time trace of nanopore bubble blockage. By running the algorithm, we have automated the detection of the nucleation point and collapse point. From these two points, 4 features are extracted. The current values at the nucleation point and collapse point gives nucleation current and collapse current respectively. The time separation between the nucleation point of a bubble and the collapse point of the previous bubble gives the waiting time for bubble nucleation. The time separation between the nucleation point and collapse point of the same bubble gives the blockage duration. (b) The heat map shows the correlation coefficient between the 4 features. (c) The histogram shows two peaks related to blockage duration, indicating two types of nanopore bubbles. The second peak which happens at a higher value of blockage duration indicates homogeneous bubbles nucleating at the pore center while the first peak at lower value of blockage duration indicates heterogenous bubbles on the cylindrical pore surface.

1. Preparing the input file

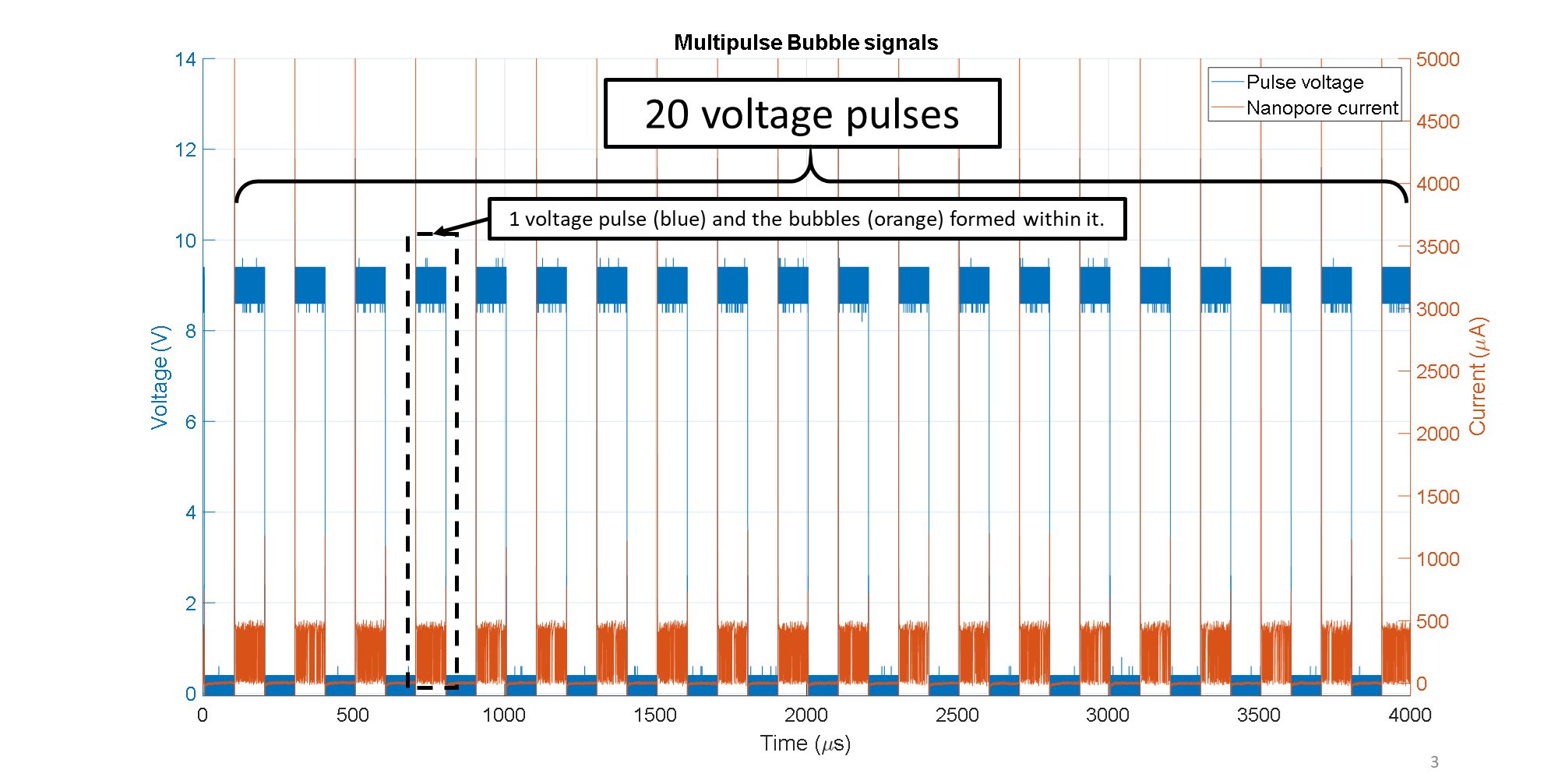


Fig.2 Multipulse bubble generation results obtained from oscilloscope. By running filtering16\_M\_1.m, the signals are filtered and divided into separate csv files corresponding to each pulse.

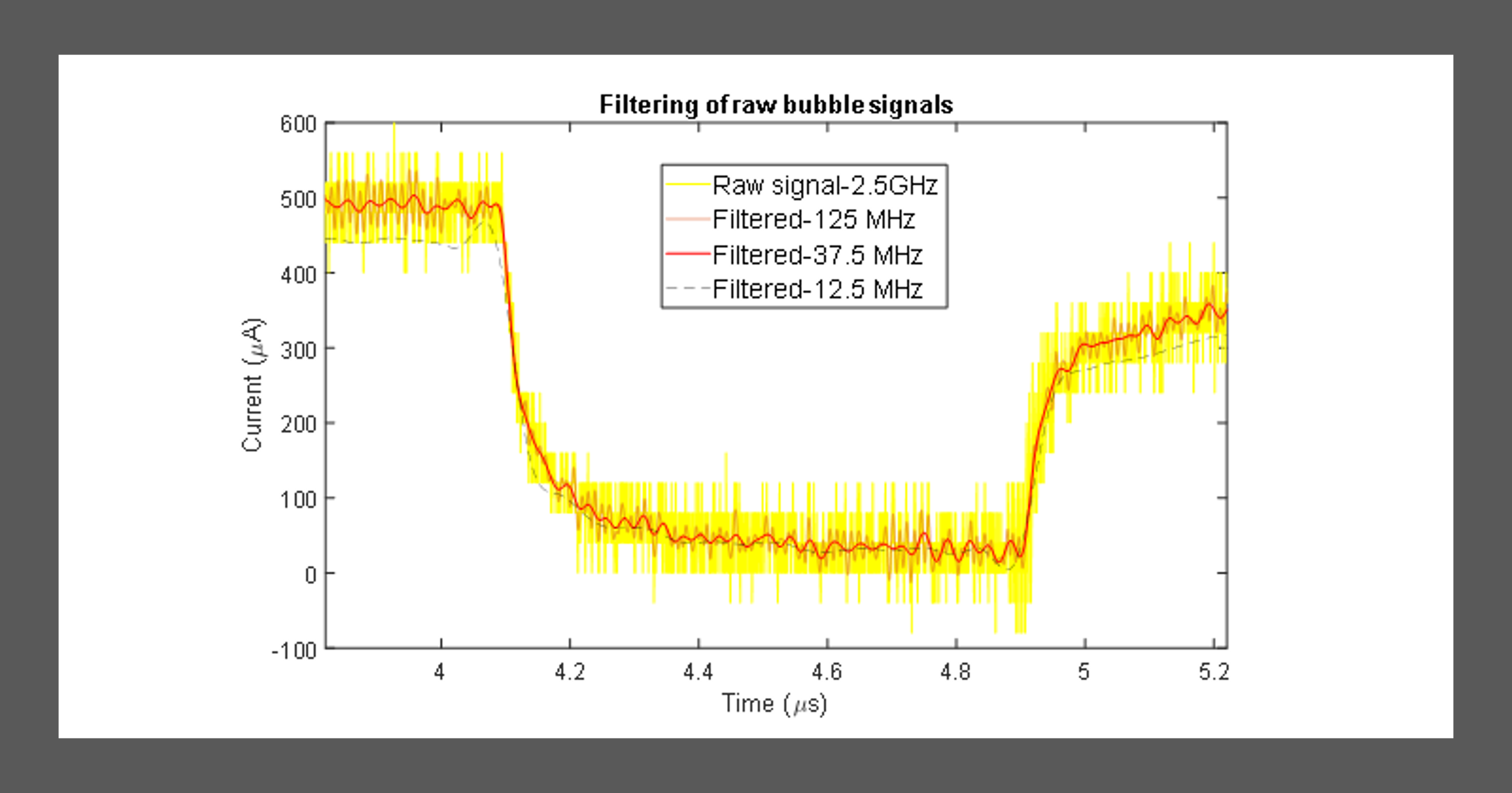


Fig.3 Filtering process in filtering16\_M\_1.m. When filtered with a low cut-off frequency~12.5 MHz (dotted black line), the shape of the signal is slightly shifted. The blockage detection algorithm can be applied either on the 12.5 MHz or 37.5 MHz signals. Parameter tuning is relatively easier for the 12.5 MHz signal as there is less noise, but some signal features are compromised.

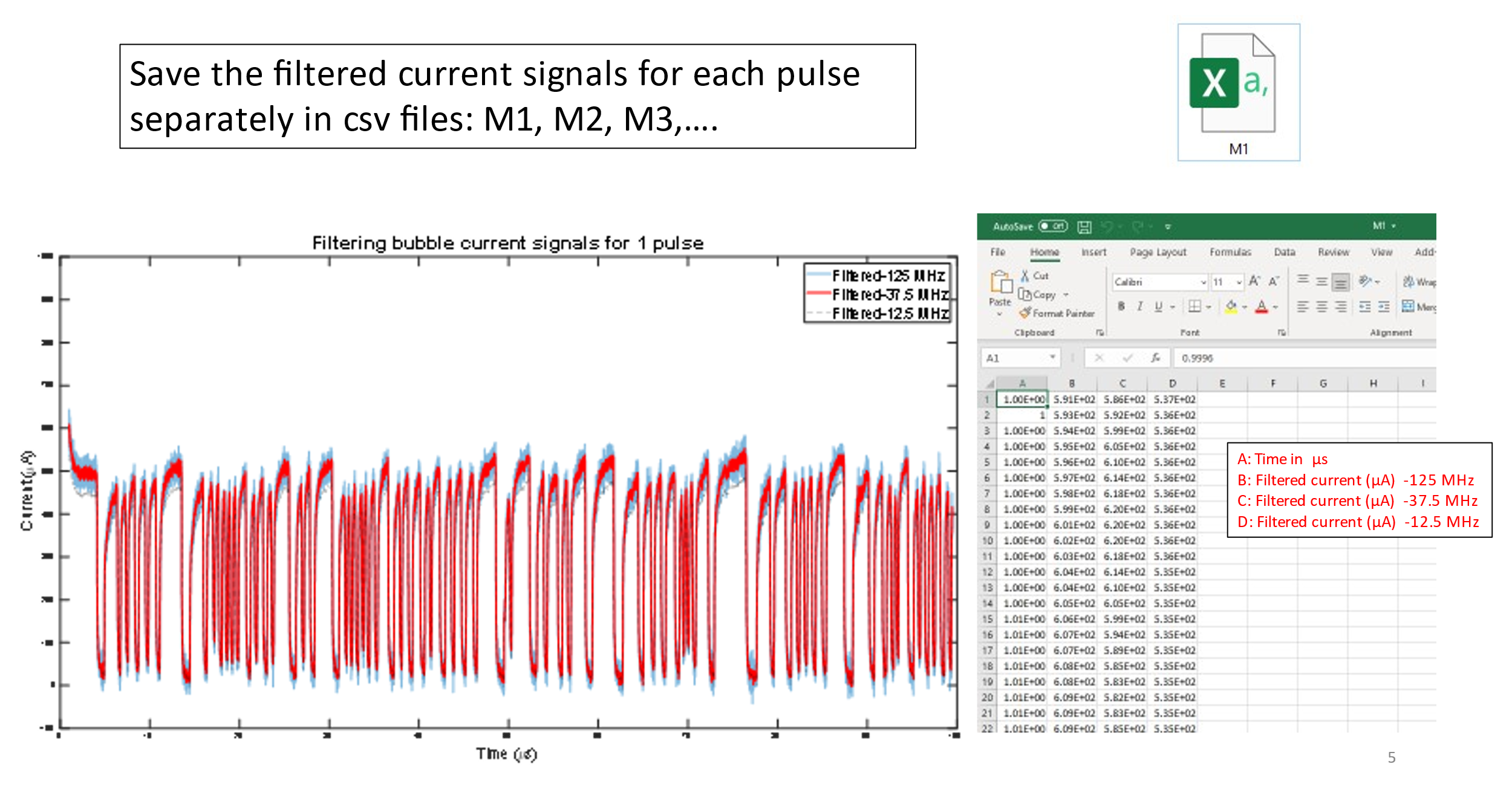


Fig.4 post filtering, the current signal filteresd at different cut-off frequencies are saved in csv files.

1. Using detectingsignals.m on bubble signals in each pulse (M1.csv)
   1. **Section 0:**

In this section, we load the csv file and set the control parameters for detection of bubbles.

counterpulse = 1;

filename = sprintf('M%i.csv',fix(counterpulse));

start = 0;

%ending = 137600;

ending = 246250;

%ending = 621150;

X1 = csvread(filename,start,0,[start,0,ending,3]);

startpt = 200\*(counterpulse-1);

endptt = 100+200\*(counterpulse-1);

T1 = 0.0004;

RawSignal = X1(:,2);

LowFilteredSignal = X1(:,3);

t = X1(:,1);

* counterpulse is the pulse number which is an integer usually between 1 to 20.
* filename is defining the input file name
* start and ending are the the row numbers in the csv files which we select for data processing. start is always zero but ending can be different. In some cases the last few hundred or thousand data rows need to be excluded as it may be an incomplete bubble at the end of the pulse. The code will not work if the whole bubble is not present.
* X1 = csvread… reads the csv data into a matrix X1.
* startpt and endpt are time points corresponding to the beginning and end of pulse.
* T1 is the time period between data points. Here it is 0.0004 microseconds.
* RawSignal is the signal filtered at 12.5 MHz and LowFilteredSignal is the signal filteres at 125 MHz. t is the time array. All of these variables are read from X1 matrix.

base1 = 150;

base2 = 350;

base12 = 200;

base15 = 250;

base18 = 300;

K = 10e5;

s1 = 950;

s2 = s1-450;

cutCurrentd = @(t) s1-base1-s2\*(1-exp(-K\*1e-6\*(t-startpt)));

cutCurrent12 = @(t) s1-base12-s2\*(1-exp(-K\*1e-6\*(t-startpt)));

cutCurrent15 = @(t) s1-base15-s2\*(1-exp(-K\*1e-6\*(t-startpt)));

cutCurrent18 = @(t) s1-base18-s2\*(1-exp(-K\*1e-6\*(t-startpt)));

cutCurrentr = @(t) s1-base2-s2\*(1-exp(-K\*1e-6\*(t-startpt)));

* cutCurrentd (red), cutCurrent12 (cyan), cutCurrent15 (black), cutCurrent12 (yellow) and cutCurrentr (green) are 4 functions of time defined to cut the RawSignal (current vs time) graph at 5 different points as shown the figure below. The parameters like base, K, s1 and s2 are defined which control the shape of these cutting functions. For different pore sizes and voltage, these parameters need to be tuned.

sigma = 5;

recede\_fac\_ini = 0.001;

order = 3;

averagingfac=20/1250;

nucpoint\_averaging\_width = 28;

* These parameters would be explained in the following sections.



Fig.5 The process of segmenting the current-time trace with the cutCurrent functions.

* 1. **Section 1:**

In this section, we use the cutCurrent functions to cut the RawSignal graph. The main goal is to make sure that we register the cut point for cutCurrentd only during the current dipping part of the bubble signal. On the other hand, the cut points during the rise part of the bubble signal needs to be registered for cutCurrent12 (cyan), cutCurrent15 (black), cutCurrent12 (yellow) and cutCurrentr (green).

Cutpt1 = zeros(1000,1);

Cutpt12 = zeros(1000,1);

Cutpt15 = zeros(1000,1);

Cutpt18 = zeros(1000,1);

Cutpt2dash = zeros(1000,1);

counter1 = 1;

counter12 = 1;

counter15 = 1;

counter18 = 1;

counter2 = 1;

i = 1;

spacer = 10;

switch1 = 0;

* The cut points arrays (e.g. Cutpt1) for the 5 cutting functions are initialized. counter1, counter12,…. are counters for the cut points for the 5 functions for different bubble signals.
* spacer is the time point shift after one cut point is detected.

while i < length(RawSignal)

if (RawSignal(i)<cutCurrentd(t(i))+sigma/2) && (RawSignal(i)>cutCurrentd(t(i))-sigma/2) && sign(RawSignal(i+200)-RawSignal(i)) < 0 && sign(RawSignal(i+50)-RawSignal(i)) < 0 && switch1 == 0

Cutpt1(counter1) = i;

counter1 = counter1+1;

i = i+spacer;

switch2 = 0;

switch1 = 1;

switch12 = 0;

switch15 = 0;

switch18 = 0;

end

if (RawSignal(i)<cutCurrent12(t(i))+sigma/2) && (RawSignal(i)>cutCurrent12(t(i))-sigma/2) && sign(RawSignal(i+spacer)-RawSignal(i)) > 0 && switch12 == 0

Cutpt12(counter1) = i;

counter12 = counter12+1;

i = i+spacer;

switch2 = 0;

switch1 = 0;

switch12 = 1;

switch15 = 0;

switch18 = 0;

end

if (RawSignal(i)<cutCurrent15(t(i))+sigma/2) && (RawSignal(i)>cutCurrent15(t(i))-sigma/2) && sign(RawSignal(i+spacer)-RawSignal(i)) > 0 && switch15 == 0

Cutpt15(counter1) = i;

counter15 = counter15+1;

i = i+spacer;

switch2 = 0;

switch1 = 0;

switch12 = 0;

switch15 = 1;

switch18 = 0;

end

if (RawSignal(i)<cutCurrent18(t(i))+sigma/2) && (RawSignal(i)>cutCurrent18(t(i))-sigma/2) && sign(RawSignal(i+spacer)-RawSignal(i)) > 0 && switch18 == 0

Cutpt18(counter1) = i;

counter18 = counter18+1;

i = i+spacer;

switch2 = 0;

switch1 = 0;

switch12 = 0;

switch15 = 0;

switch18 = 1;

end

if (RawSignal(i)<cutCurrentr(t(i))+sigma/2) && (RawSignal(i)>cutCurrentr(t(i))-sigma/2) && sign(RawSignal(i+1)-RawSignal(i)) > 0 && switch2 == 0

Cutpt2dash(counter2) = i;

counter2 = counter2+1;

i = i+spacer;

switch2 = 1;

switch1 = 0;

switch12 = 0;

switch15 = 0;

switch18 = 0;

end

i = i+1;

end

Cutpt2dash(Cutpt2dash==0)=[];

Cutpt1(Cutpt1==0)=[];

Cutpt12(Cutpt12==0)=[];

Cutpt15(Cutpt15==0)=[];

Cutpt18(Cutpt18==0)=[];

* In this while loop, the counter i is run across the time points in the pulse. For a given value of i, if the RawSignal current value lies within the limits of cutCurrentd(t(i))-sigma/2) and cutCurrentd(t(i))+sigma/2), that point becomes the effective cut point. To make sure that the cutCurrentd cuts the bubble signals only when the current is falling an additional requirement of sign(RawSignal(i+200)-RawSignal(i)) < 0 is imposed. Also, there is a switch1 which should be 0. Once such a value of i is detected, it is stored in the Cutpt1 array.
* i = i+spacer; is used to move the i counter by 10 points. So that we move out of the [cutCurrentd(t(i))-sigma/2), cutCurrentd(t(i))+sigma/2)] range and 2 Cutpt1 for the same bubble signal is not recorded. **sigma** is the tolerance.
* Cutpt1, Cutpt12, Cutpt15, Cutpt18, Cutpt2dash contain the index of the points where the functions cut the current trace.
* switch2 = 0; switch1 = 1; switch12 = 0; switch15 = 0; switch18 = 0; are set which make sure that we do not enter this if statement again before the other cut points corresponding to the other functions are detected.
* It should be noted that not bubble signals would be cut by all the rise curve cutting functions (cutCurrent12 (cyan), cutCurrent15 (black), cutCurrent12 (yellow) and cutCurrentr (green)) as there are bubble signals of different magnitudes of current dips.
* The excess array elements in the Cutpt1, Cutpt12,… variables are deleted after this loop ends.
  1. **Section 2:**

In the previous section, we got 4 or less cut points (Cutpt12, Cutpt15, Cutpt18 and Cutpt2dash) for the current rising part of bubble signals. From these 4 points we need to select only one point. Our desire is to choose one of these 4 points, which is close but not too close to the collapse point. Let these points be stored in Cutpt2dashdash.

lengthfit = zeros(length(Cutpt1),1)+70;

Cutpt2dashdash = zeros(length(Cutpt1),1);

for i=1:length(Cutpt1)

if (Cutpt2dash(i) == min([Cutpt12(i),Cutpt15(i),Cutpt18(i), Cutpt2dash(i)])) && (RawSignal(Cutpt2dash(i)) > min([RawSignal(Cutpt1(i):Cutpt2dash(i))])+50)

Cutpt2dashdash(i) = Cutpt2dash(i);

elseif Cutpt18(i) == min([Cutpt12(i),Cutpt15(i),Cutpt18(i)]) && RawSignal(Cutpt18(i)) > min([RawSignal(Cutpt1(i):Cutpt18(i))])+50

Cutpt2dashdash(i) = Cutpt18(i);

if (Cutpt2dash(i) ~= min([Cutpt12(i),Cutpt15(i),Cutpt18(i), Cutpt2dash(i)]))

b = [Cutpt2dash(1:i-1);0;Cutpt2dash(i:end)];

Cutpt2dash = b;

display('cutpt2dash miss')

display(i)

end

elseif Cutpt15(i) == min(Cutpt12(i),Cutpt15(i)) && RawSignal(Cutpt15(i)) > min([RawSignal(Cutpt1(i):Cutpt15(i))])+50

Cutpt2dashdash(i) = Cutpt15(i);

if (Cutpt2dash(i) ~= min([Cutpt12(i),Cutpt15(i),Cutpt18(i), Cutpt2dash(i)]))

b = [Cutpt2dash(1:i-1);0;Cutpt2dash(i:end)];

Cutpt2dash = b;

display('cutpt2dash miss')

display(i)

end

if Cutpt18(i) ~= min([Cutpt12(i),Cutpt15(i),Cutpt18(i)])

b = [Cutpt18(1:i-1);0;Cutpt18(i:end)];

Cutpt18 = b;

display('cutpt18 miss')

display(i)

end

else

Cutpt2dashdash(i) = Cutpt12(i);

[minC,minindex] = min([RawSignal(Cutpt1(i):Cutpt12(i))]);

lengthfit(i) = ceil(0.5\*(length([RawSignal(Cutpt1(i):Cutpt12(i))])-minindex));

if (Cutpt2dash(i) ~= min([Cutpt12(i),Cutpt15(i),Cutpt18(i), Cutpt2dash(i)]))

b = [Cutpt2dash(1:i-1);0;Cutpt2dash(i:end)];

Cutpt2dash = b;

display('cutpt2dash miss')

display(i)

end

if Cutpt18(i) ~= min([Cutpt12(i),Cutpt15(i),Cutpt18(i)])

b = [Cutpt18(1:i-1);0;Cutpt18(i:end)];

Cutpt18 = b;

display('cutpt18 miss')

display(i)

end

if Cutpt15(i) ~= min(Cutpt12(i),Cutpt15(i))

b = [Cutpt15(1:i-1);0;Cutpt15(i:end)];

Cutpt15 = b;

display('cutpt15 miss')

display(i)

end

end

end

Cutpt2dash = Cutpt2dashdash;

* In the previous section, we mentioned that all rise cutting functions would not cut all bubble signals. But the Cutpt1 is designed to cut all target bubble signals.
* So, we try to find the desired Cutpt2dashdash by running a counter within the array of Cutpt1.
* For each value of Cutpt1, we try to find the minimum of Cutpt12, Cutpt15, Cutpt18, Cutpt2dash and the minimum of them is 50 microamperes above the minimum current point for that bubble.
  + Case 1: For the given bubble signal all of them exist, and current value at Cutpt2dash is 50 microamperes above the minimum current point for that bubble. In this case, Cutpt2dashdash = Cutpt2dash and the loop will move onto the next value of Cutpt1.
  + Case 2: For the given bubble signal all of them exist, *but* current value at Cutpt2dash is not 50 microamperes above the minimum current point for that bubble. In this case, we will check whether Cutpt18 satisfies the same conditions. If not, we will move onto Cutpt15 and so on.
  + Case 3: For the given bubble signal all of them *do not* exist. This will be particularly true for small hetero bubbles which have high values of collapse currents. For example, if for i=4, (4th bubble), only Cutpt12, Cutpt15, Cutpt18 exist, then the time index stored in Cutpt2dash(4) will actually correspond to the i=5, (5th bubble) or later bubble. So, Cutpt2dash(4)>[ Cutpt12(4), Cutpt15(4), Cutpt18(4)] and the first if statement wont be satisfied. In this case also, the first elseif statement will be activated and we will check whether Cutpt18 exists or not and it is 50 microamperes above the minimum current point for that bubble. If not, we will check for Cutpt15 and so on.

When a Cutpt is missed (ex. Cutpt2dash(4)) and we detect it in this section, we remedy this error by inserting a dummy value of ‘0’ at this point:

b = [Cutpt2dash(1:i-1);0;Cutpt2dash(i:end)];

Cutpt2dash = b;

display('cutpt2dash miss')

* + The same process is applied for other Cutpts as well when they are found to not exist for a given Cutpt1.
  + Case 4: Only Cutpt12 exists. The selection process is the same as described earlier. However, in this case we also store the half of the time index length of signal from Cutpt12 to the minimum current point for the bubble inside the array, **lengthfit**. We will use this array later in Section 4.
* After the array of Cutpt2dashdash is finalized, it is stored in Cutpt2dash.
  1. **Section 3:**

Cutpt2 = zeros(length(Cutpt2dash),1);

%%%

%recede

L\_array = Cutpt2dash-Cutpt1;

l\_ini = max(L\_array);

kappa = recede\_fac\_ini/power(l\_ini,order);

for i=1:length(Cutpt2dash)

recede\_fac = 1-kappa\*power(Cutpt2dash(i)-Cutpt1(i),order);

Cutpt2(i) = floor((1-recede\_fac)\*Cutpt1(i)+recede\_fac\*Cutpt2dash(i));

end

* In this section, we will recede the Cutpt2dash time indices towards the real collapse point (Fig.6).
* Let the receded point array be Cutpt2.
* First we store the length of each bubble inside L\_array. l\_ini is the maximum bubble lifetime.
* Cutpt2 or is calculated by a weighted mean of (Cutpt1) and (Cutpt2dash) as , where the weighing factor is also related to the duration of bubble lifetime as . is **recede\_fac\_ini** and is **order,** which are tunable parameters defined earlier. In this formulation, the weighing factor is non-linear and related to the bubble size as well.



Fig.6 The process of receding the Cutpt2dash points. At the end of Section 3, we have obtained Cutpt2dash, but we recede the points towards the real collapse point so that when we fit a straight line to the current rise curve, it better approximates the slope near the collapse point.

* 1. **Section 4:**

In this section, we will use Cutpt1 and Cutpt2 to estimate the dip point (dippt) or nucleation point.

curvept1 = zeros(length(Cutpt1),1);

curvept2 = zeros(length(Cutpt2),1);

curvept1(1) = Cutpt1(1)-500;

for i = 1:length(Cutpt1)

curvept2(i) = Cutpt1(i);

curvept1(i+1) = Cutpt2(i);

end

dippt = zeros(length(curvept1)-1,1);

locationdippt = zeros(length(curvept1)-1,1);

% tolerance = 0.1;

factor = 0.01;

for i=1:length(curvept1)-1

% for i=1:1

ptssavet = t(curvept1(i):curvept2(i));

% if length(ptssavet)>50

ptssaveRS = RawSignal(curvept1(i):curvept2(i));

% f2 = fit(ptssavet,ptssaveRS,'fourier6');

% ptsonf2 = f2(ptssavet);

c = polyfit(ptssavet(length(ptssavet)-lengthfit(i):length(ptssavet)),ptssaveRS(length(ptssavet)-lengthfit(i):length(ptssavet)),1);

tolerance = factor + abs((ptssaveRS(length(ptssavet))-c(2))/c(1)-ptssavet(length(ptssavet)));

% d = zeros(length(ptssavet),1);

d = tolerance - factor;

j = length(ptssavet);

marker = 1;

disttt = zeros(length(ptssavet),1);

while j >= 1 && marker == 1

% d = abs(ptssavet(j)-c(1)\*ptssaveRS(j)-c(2))/(sqrt(power(c(1),2)+1));

d = abs((ptssaveRS(j)-c(2))/c(1)-ptssavet(j));

disttt(j) = d;

if d > tolerance

dippt(i) = ptssavet(j-1);

marker = 0;

end

j = j-1;

end

arr = find(t==dippt(i));

locationdippt(i) = min(arr);

if i<4 && i>2

figure

hold on

plot(t,RawSignal)

hold on

plot(ptssavet,c(1)\*ptssavet+c(2))

hold on

plot(ptssavet(length(ptssavet)),ptssaveRS(length(ptssavet)),'\*b')

hold on

plot(dippt(i),RawSignal(locationdippt(i)),'\*k')

hold off

hold on

plot(t(Cutpt1),RawSignal(Cutpt1),'\*r')

hold on

plot(t(Cutpt2),RawSignal(Cutpt2),'\*g')

hold off

ylim([50 600])

xlim([t(Cutpt2(i-1)) t(Cutpt1(i+1))])

ylabel('Current (\muA)')

xlabel('Time (\mus)')

title('Dip point for 3rd bubble')

end

end

* First curvept1 and curvept2 are defined which are namely the Cutpt2 for the previous bubble and Cutpt1 for the current bubble (Fig.7a). For the first bubble, curvept1 is 500 points before Cutpt1.
* Now a loop is run for each bubble, where ptssavet and ptssaveRS contain the time co-ordinates and current co-ordinates between curvept1 and curvept2.
* Now a straight line (c = polyfit(…….) is fitted to a section of ptssaveRS curve, (starting from curvept2 moving backwards by **lengthfit(i)** points). For all bubbles except those cut only by cutCurrent12, lengthfit(i) = 70.
* Now, we calculate the horizontal distance (disttt) of the current curve from the straight line, c. Tolerance is defined by the sum of distance from Cutpt1 and factor.
* We run a while loop starting from Cutpt1, moving backwards towards curvept1 and calculate disttt (Fig7b). When disttt exceeds the tolerance, we name that point as Dippt or nucleation point and move out of the while loop.

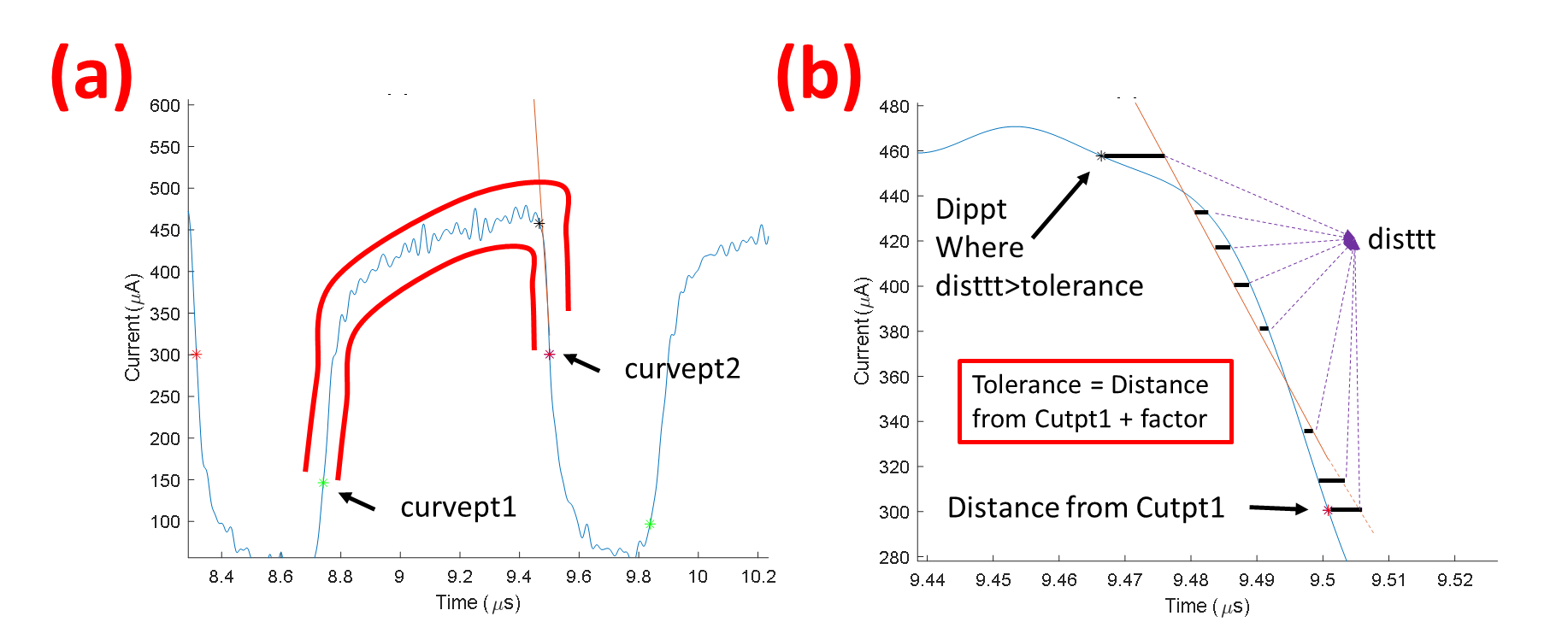


Fig.7 (a) curvept1 and curvept2. (b) Methodology for dippt calculation. The orange line is the straight line fitted to the falling current region near the Cutpt1.

* 1. **Section 5:**

risept = zeros(length(curvept1)-1,1);

locationrisept = zeros(length(curvept1)-1,1);

factor1 = 0.001;

for i=1:length(curvept1)-1

ptssavet = t(curvept2(i):curvept1(i+1));

ptssaveRS = RawSignal(curvept2(i):curvept1(i+1));

lengthfit1 = averagingfac\*length(ptssavet);

c = polyfit(ptssavet(length(ptssavet)-lengthfit1:length(ptssavet)),ptssaveRS(length(ptssavet)-lengthfit1:length(ptssavet)),1);

tolerance = factor + abs((ptssaveRS(length(ptssavet))-c(2))/c(1)-ptssavet(length(ptssavet)));

d = tolerance - factor;

j = length(ptssavet);

marker = 1;

disttt = zeros(length(ptssavet),1);

while j >= 1 && marker == 1

d = abs((ptssaveRS(j)-c(2))/c(1)-ptssavet(j));

disttt(j) = d;

if d > tolerance

risept(i) = ptssavet(j+1);

marker = 0;

end

j = j-1;

end

arr = find(t==risept(i));

locationrisept(i) = min(arr);

if i<4 && i>2

figure

hold on

plot(t,RawSignal)

hold on

plot(ptssavet,c(1)\*ptssavet+c(2))

hold on

plot(ptssavet(length(ptssavet)),ptssaveRS(length(ptssavet)),'\*b')

hold on

plot(risept(i),RawSignal(locationrisept(i)),'\*k')

hold off

hold on

plot(t(Cutpt1),RawSignal(Cutpt1),'\*r')

hold on

plot(t(Cutpt2),RawSignal(Cutpt2),'\*g')

hold off

ylim([50 250])

xlim([t(Cutpt2(i-1)) t(Cutpt1(i+1))])

ylabel('Current (\muA)')

xlabel('Time (\mus)')

title('Rise point for 3rd bubble')

end

end

* The method of calculation of Risept is exactly similar to Dippt. Here we caculate the fitting length, lengthfit1 by multiplying a factor (averagingfac) as averagingfac\*length(ptssavet)

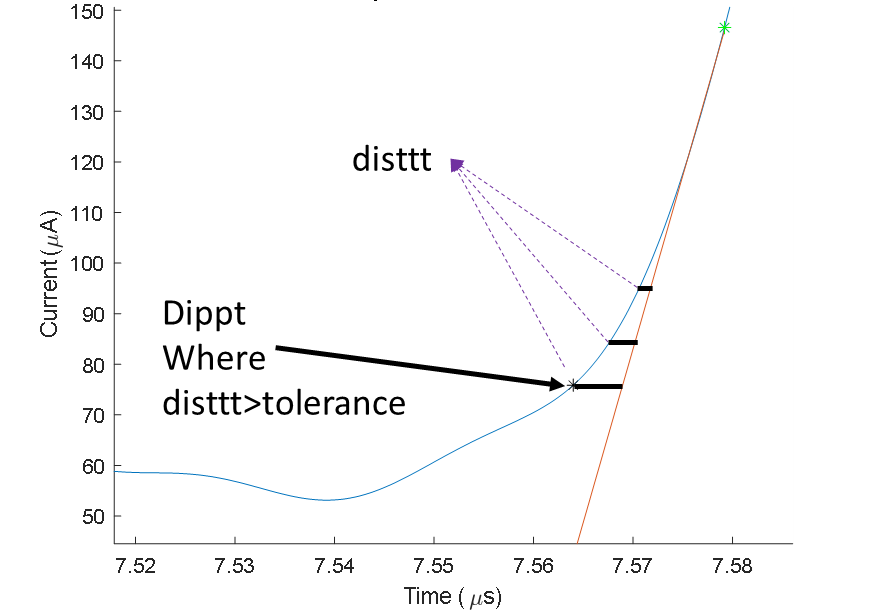


Fig.8 Methodology for risept calculation