In [5]:

```python
'''
JupyterLab Notbook (2.2.6) for Phython code accompaning the publication:

Large, stable spikes exhibit differential broadening in excitatory and inhibitory neocortical boutons
Andreas Ritzau-Jost, Timur Tsintsadze, Martin Krueger, Jonas Ader, Ingo Bechmann, Jens Eilers, Boris Barbour, Stephen M. Smith, and Stefan Hallermann
Cell Reports 2021 34:108612. doi: 10.1016/j.celrep.2020.108612

In short, the semi-interactive script allows to determine the peak-to-peak duration of extracellular
recorded loose-seal currents from boutons. Action potentials were elicited by a paired whole-cell recording
at the soma. The script allows averaging of bouton loose-seal currents aligned to the first current peak.

This notebook contains seven cells which are shortly explained in the following.
At the beginning of each cell more detailed comments can be found.
Additionally, you can find an example plot underneath each cell where data is plotted.


The provided example data "testData_100Hz_train.dat" illustrate the function of the program for a 100Hz train of 20 action potentials.
However, the code allows also to analyse single action potentials, for testing, a .dat-file containg single APs (testData_singleAPs.dat) is provided.
To analyse single APs nRep = 2 must be set, apart from this, the analysis follows the same steps.

Cell#1: -Selection .dat file created by PATCHMASTER and respective series/sweeps from within the file
        -Executing the first cell will create an array containing selected data
Cell#2: Selection of intervals for subsequent fitting of current peaks
Cell#3: Fitting of noise preceding current peaks with similar intervals
Cell#4: Baseline determination for each individual stimulus during train stimulation (accounting for systematic changes in action potential delay during the train)
Cell#5: Alignment of each stimulus to negative current peak and average train calculation
Cell#6: Creation of arrays containing results (e.g. time-interval between peaks)
Cell#7: Creation of .txt-files of result-arrays

Jonas Ader and Stefan Hallermann
December 2020

For more information contact hallermann@medizin.uni-leipzig.de
'''

import heka_reader
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal import butter,filtfilt
from scipy.interpolate import UnivariateSpline
import decimal
from mpl_axes_aligner import align
import subprocess

%matplotlib

decimal.getcontext().prec=16


'''selecting .dat-file and series/sweeps within the selected .dat-file'''
# loading .dat file
bundle = heka_reader.Bundle('testData_100Hz_train.dat')

#defining series and within each series sweeps that will be imported
firstSeries = 1                          #for choosing only one series: firstSeries=lastSeries
lastSeries = 4

firstSweep = 1                           #for choosing only one sweep: firstSweep=lastSweep
lastSweep = 1

startTime = 0.098                        #starting time point of train (first stimulus) in s
freq = 0.01                              #period (freq^-1)
nRep = 20                                #number of stimuli in each train

cutoffAcoord, cutoffVcoord = 8500,8500  #Lowpass-Filter (Hz)


'''
structure of created array:

- from each selected sweep an object of the class VmonAdc (for attributs see class description) is created containing one train/stimulation
- each object then is cut into multiple objects of the class VmonAdc according to "freq" and "nRep" provided above
- each new object contains one individual train event, e.g. a sweep with a train of 20 stimuli is cut into 20 objects, the first object contains the first stimulus, the second object contains the  second stimulus an so on
- when several sweeps are selected, an array of these sweeps is created
- the number of lines in this array equals the number of sweeps
- the number of columns in this array equals the number of stimuli per train
 -> each element of the array is an object of VmonAdc and contains one stimulus
'''


class VmonAdc (object):
    '''

    each object of the class VmonAdc contains data of a HEKA Patchmaster .dat file


    Attributes
    ----------
    XInterval : float
        distance between each sample point of the .dat file in s

    Xcoord : list
        list of timeValues from the recording

    Vcoord : list
        list of voltageValues from the soma-AP

    Acoord : list
        list of currentValues from AP-evoked currents recorded from the bouton

    Interval : array
        [list(timeValues)]
        [list(currentValues)]

        contains fitted currentValues and according timeValues for manually choosen interval
        when creating object, before defining the relevant interval, Interval is empty

    Datapoints : array
        [timeValueMin][currentValueMin]
        [timeValueMax][currentValueMax]

        contains lowest and highest currentValues from Interval, highest point needs to be after lowest point

    smoothingFactor : float
        look at description from scipy.interpolate.UnivariateSpline

    '''
    def __init__(self, Xinterval, Xcoord, Vcoord, Acoord):
        self.Xinterval = Xinterval

        self.Xcoord = Xcoord
        self.Vcoord = Vcoord
        self.Acoord = Acoord

        self.Interval = []
        self.Datapoints = []

        self.smoothingFactor = 0.001


    def getDatapoint(self, time):

        #returns index of datapoint at time, relative to startpoint of sweep

        datapoint = int(time/self.Xinterval)
        return datapoint


    def getRelativeDatapoint(self, time):

        #returns index of datapoint at time, relative to startpoint of train

        relativeTime = time - self.Xcoord[0]
        datapoint = int(relativeTime/self.Xinterval)
        return datapoint


    def setSmoothingFactor(self, smoothingFactor):
        self.smoothingFactor = smoothingFactor


    def univariateSplineFit(self, startInterval, endInterval):

        #calculates spline fit for the current values in the choosen interval and copies them, together with according time values, into Interval
        #additionally determines lowest and highest points and copies them, together with according time values, into Datapoints

        #calculating spline fit
        uniSplineData = UnivariateSpline(self.Xcoord[startInterval:endInterval+1],\
                                         self.Acoord[startInterval:endInterval+1])

        #defining smoothing factor
        uniSplineData.set_smoothing_factor(self.smoothingFactor)

        #copying data into Interval
        self.Interval = np.array([self.Xcoord[startInterval:endInterval+1],\
                                  uniSplineData(self.Xcoord[startInterval:endInterval+1])])

        #determining low/high point and copying it into Datapoints
        self.Datapoints = np.array([[self.Interval[0][np.argmin(self.Interval[1])],self.Interval[1][np.argmin(self.Interval[1])]],\
                                    [self.Interval[0][np.argmax(self.Interval[1][np.argmin(self.Interval[1]):])+len(self.Interval[1])-len(self.Interval[1][np.argmin(self.Interval[1]):])],self.Interval[1][np.argmax(self.Interval[1][np.argmin(self.Interval[1]):])+len(self.Interval[1])-len(self.Interval[1][np.argmin(self.Interval[1]):])]]])


    def butterLowpassFilter(self, cutoffAcoord, cutoffVcoord, order=4):

        #filtering data using Butterworth lowpass filter

        normalCutoffAcoord = cutoffAcoord / (0.5*(self.Xinterval**-1))
        normalCutoffVcoord = cutoffVcoord / (0.5*(self.Xinterval**-1))

        bAcoord, aAcoord = butter(order, normalCutoffAcoord, btype='low', analog=False)
        self.Acoord = filtfilt(bAcoord, aAcoord, self.Acoord)

        bVcoord, aVcoord = butter(order, normalCutoffVcoord, btype='low', analog=False)
        self.Vcoord = filtfilt(bVcoord, aVcoord, self.Vcoord)

    def setLowZero(self):

        #defining lowPoint from Datapoints as t=0

        indexLowBefore = int(np.argwhere(self.Xcoord == self.Datapoints[0][0]))

        startInterval = int(np.argwhere(self.Xcoord == self.Interval[0][0]))
        endInterval = int(np.argwhere(self.Xcoord == self.Interval[0][-1]))

        self.Xcoord = []

        for coord in range(0,len(self.Vcoord)):
            self.Xcoord.append((coord-indexLowBefore)*self.Xinterval)

        self.univariateSplineFit(startInterval, endInterval)

    def baselineCorrection(self, averageDurStart=0, averageDurEnd=0.05):

        #defining baseline by calculating average of YValues from averageDurStart till averageDurEnd

        if len(self.Interval) != 0:
            startInterval = int(np.argwhere(self.Xcoord == self.Interval[0][0]))
            endInterval = int(np.argwhere(self.Xcoord == self.Interval[0][-1]))

        averageListV = self.Vcoord[self.getRelativeDatapoint(averageDurStart):self.getRelativeDatapoint(averageDurEnd)]
        averageV =sum(averageListV)/len(averageListV)

        for index in range(0, len(self.Vcoord)):
            self.Vcoord[index] = self.Vcoord[index]-averageV

        averageListA = self.Acoord[self.getRelativeDatapoint(averageDurStart):self.getRelativeDatapoint(averageDurEnd)]
        averageA =sum(averageListA)/len(averageListA)

        for index in range(0, len(self.Acoord)):
            self.Acoord[index] = self.Acoord[index]-averageA

        try:
            self.univariateSplineFit(startInterval, endInterval)
        except:
            pass


#following functions create objects of VmonAdc and arrange them in an array


def makeVmonAdc(group, series, sweep):
    '''

    creating object of VmonAdc from choosen sweep of loaded .dat file
    additionally interpolating data linearly, therby increasing number of datapoints ten-fold

    '''

    #getting data from .dat
    trace = bundle.pul[group][series][sweep][0]
    Xinterval = float(decimal.Decimal(trace.XInterval)*decimal.Decimal(0.1))

    X = np.linspace(trace.XStart, trace.XInterval*(trace.DataPoints-1), trace.DataPoints, endpoint=True)

    dataVmon = bundle.data[group, series, sweep, 0]
    dataAdc = bundle.data[group, series, sweep, 2]

    Xcoord = []
    for index in range(0,trace.DataPoints*10-9):
        Xcoord.append(index*Xinterval)

    #Interpolation
    Vcoord = np.interp(Xcoord, X, dataVmon)
    Acoord = np.interp(Xcoord, X, dataAdc)

    #creating an VmoAdc object ("sweep") containing all data of choosen sweep
    sweep = VmonAdc(Xinterval, Xcoord, Vcoord, Acoord)
    sweep.baselineCorrection()
    sweep.butterLowpassFilter(cutoffAcoord, cutoffVcoord)

    return sweep

def cutSweep(sweep, startTime, freq, nRep):
    '''

    cutting the object "sweep" into various objects
    creating list of these objects, each object containing data from one stimulation of a train

    '''
    cutSweep = []

    startDatapoint = sweep.getDatapoint(startTime)
    period = sweep.getDatapoint(freq)

    cutSweep.append(VmonAdc(\
                sweep.Xinterval,\
                sweep.Xcoord[startDatapoint:startDatapoint+period+1],\
                sweep.Vcoord[startDatapoint:startDatapoint+period+1],\
                sweep.Acoord[startDatapoint:startDatapoint+period+1]))
    for x in range(1,nRep):
        cutSweep.append(VmonAdc(\
                sweep.Xinterval,\
                sweep.Xcoord[startDatapoint+x*period:startDatapoint+(x+1)*period+1],\
                sweep.Vcoord[startDatapoint+x*period:startDatapoint+(x+1)*period+1],\
                sweep.Acoord[startDatapoint+x*period:startDatapoint+(x+1)*period+1]))

    return cutSweep

def makeArraySweep(firstSeries, lastSeries, firstSweep, lastSweep, group=0, startTime=startTime, freq=freq, nRep=nRep):
    '''

    Returns
    -------
    arraySweep : array
        array of objects, each row equals singleTrace (single-column-array) or train (multi-column-array)

    '''
    arraySweep = []

    for xSeries in range(firstSeries, lastSeries+1):
        for xSweep in range(firstSweep, lastSweep+1):
            arraySweep.append(cutSweep(makeVmonAdc(group, xSeries, xSweep), startTime, freq, nRep))

    arraySweep = np.asarray(arraySweep)

    return arraySweep


def plotObjectsArray(arraySweep, startRow, endRow, startColumn, endColumn, ):
    '''

    ----------
    create figure before using this function
    e.g. fig = plt.figure(figsize=(10,7))


    ----------
    function plots choosen objects after clearing previously created figure

    '''

    fig.clear()

    axs1 = fig.add_subplot()
    axs1.set_xlabel('time (s)')
    axs1.set_ylabel('mV', color='black')
    axs1.tick_params(axis='y', labelcolor='black')

    axs2 = axs1.twinx()
    axs2.set_ylabel('pA', color='blue')
    axs2.tick_params(axis='y', labelcolor='blue')

    maxAcoords = []
    minVcoords = []
    for row in range(startRow, endRow):
        for column in range(startColumn, endColumn):
            axs1.plot(arraySweep[row, column].Xcoord,\
                      arraySweep[row, column].Vcoord,\
                      color="black", linewidth=1.0, linestyle="-")

            axs2.plot(arraySweep[row, column].Xcoord,\
                      arraySweep[row, column].Acoord,\
                      color="blue", linewidth=1.0, linestyle="-")

            if len(arraySweep[row, column].Interval) != 0:
                axs2.plot(arraySweep[row, column].Interval[0],\
                          arraySweep[row, column].Interval[1],\
                          color="green", linewidth=1.0, linestyle="-")
                axs2.plot(arraySweep[row, column].Datapoints[0][0],\
                          arraySweep[row, column].Datapoints[0][1],\
                          "o", color="black")
                axs2.plot(arraySweep[row, column].Datapoints[1][0],\
                          arraySweep[row, column].Datapoints[1][1],\
                          "o", color="black")

            maxAcoords.append(max(arraySweep[row, column].Acoord))
            minVcoords.append(min(arraySweep[row, column].Vcoord))

    align.yaxes(axs1, min(minVcoords), axs2, max(maxAcoords)*1.1, None)
    fig.tight_layout()

    plt.draw()


def alignLow(arraySweep):
    '''

    aligning various trains by inward-peak -respectively AP.Datapoints[0][0]-

    calculating mean current value of each aligend datapoint

    Returns
    -------
    aligneLow[0] : int
        lenght of sample-interval in s

    aligneLow[1] : 2D-list
        list of Xcoords for each AP of train
        -> time

    aligneLow[2] : 2D-list
        list of Vcoords for each AP of train
        -> voltage

    aligneLow[3] : 2D-list
        list of Acoords for each AP of train
        -> current


    '''
    countXcoords = []

    averagedXcoordsListLow = []
    averagedVcoordsListLow = []
    averagedAcoordsListLow = []

    #selecting each column of arraySweep individually, to align them by Inward-Peak and then calculating the average
    #column-index 0 represents first stimulation, column-index 1 represents second stimulation ...
    for column in range(0, arraySweep.shape[1]):
        listLow = []

        columnSweep = arraySweep[:,column]      #columnSweep contains each AP from previously selected column of arraySweep
        for AP in columnSweep:
            listLow.append(AP.Datapoints[0][0]) #creating list of time values of Inward-Peak from each AP in selected column

        #determining earliest and latest Inward-Peak, relative to startPoint of the stimulus, to know how APs need to be cut
        XcoordMinLow = min(listLow)
        XcoordMaxLow = max(listLow)

        #adjusting each list to same length for subsequent alignment
        #startPoint of aligned Acoord/Vcoord is timePoint of Inward-Peak minus timePoint of earliest Inward-Peak
        #endPoint of aligned Acoord/Vcoord is timePoint of Inward-Peak plus distance between latest Inward-Peak and APEndPoint
        alignedVcoords = []
        alignedAcoords = []
        for AP in columnSweep:
            alignedVcoords.append(AP.Vcoord[int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0]))-int(np.argwhere(AP.Xcoord == XcoordMinLow)):\
                                            int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0]))+len(AP.Xcoord)-int(np.argwhere(AP.Xcoord == XcoordMaxLow))])
            alignedAcoords.append(AP.Acoord[int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0]))-int(np.argwhere(AP.Xcoord == XcoordMinLow)):\
                                            int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0]))+len(AP.Xcoord)-int(np.argwhere(AP.Xcoord == XcoordMaxLow))])
        alignedVcoords = np.asarray(alignedVcoords)
        alignedAcoords = np.asarray(alignedAcoords)

        #calculating average of aligned APs
        averagedXcoords = []
        averagedVcoords = []
        averagedAcoords = []
        #for time and voltage values
        for column in range(0, alignedVcoords.shape[1]):
            averagedXcoords.append(arraySweep[0,0].Xinterval*column+arraySweep[0,0].Xinterval*(len(countXcoords)-1))

            columnAlignedVcoords = alignedVcoords[:,column]
            averagedVcoord = np.sum(columnAlignedVcoords)/len(columnAlignedVcoords)

            averagedVcoords.append(averagedVcoord)

        #for current values
        for column in range(0, alignedAcoords.shape[1]):
            columnAlignedAcoords = alignedAcoords[:,column]
            averagedAcoord = np.sum(columnAlignedAcoords)/len(columnAlignedAcoords)

            averagedAcoords.append(averagedAcoord)

        countXcoords.extend(averagedXcoords)

        #adding averaged values (time, voltage, current) of each column from arraySweep to the same list
        #thereby creating a 2D-list (list made out of lists) representing the averaged train
        averagedXcoordsListLow.append(averagedXcoords)
        averagedVcoordsListLow.append(averagedVcoords)
        averagedAcoordsListLow.append(averagedAcoords)

    return arraySweep[0,0].Xinterval, averagedXcoordsListLow, averagedVcoordsListLow, averagedAcoordsListLow


def setSmoothingFactorArray(arraySweep, smoothingFactor):
    for row in range(0, arraySweep.shape[1]):
        for column in range(0, arraySweep.shape[0]):
            arraySweep[column,row].setSmoothingFactor(smoothingFactor)

def setLowZeroArray(arraySweep):
    for row in range(0, arraySweep.shape[1]):
        for column in range(0, arraySweep.shape[0]):
            arraySweep[column,row].setLowZero()

def setBaselineCorrectionBeforeIntervalsArray(arraySweep, beforeIntervalStart, beforeIntervalEnd):
    '''

    defining baseline for each fitted Interval
    from beforeIntervalStart(time in s before start of interval) to beforeIntervalEnd(time in s before start of interval)
    -> define interval where mean current value is calculated

    '''
    for row in range(0, arraySweep.shape[1]):
        for column in range(0, arraySweep.shape[0]):
            startBaseline = arraySweep[column,row].Interval[0,0]-beforeIntervalStart
            endBaseline = arraySweep[column,row].Interval[0,0]-beforeIntervalEnd

            if startBaseline < arraySweep[column,row].Xcoord[0]:
                print ('Interval too big')
                print ('maxInterval:',arraySweep[column,row].Interval[0,0]-arraySweep[column,row].Xcoord[0])
            else:
                arraySweep[column,row].baselineCorrection(averageDurStart=startBaseline,\
                                                          averageDurEnd=endBaseline)

def makeAlignedArraySweep(outAlign):
    '''

    creating array with aligend data

    '''
    listSweep = []

    for AP in range(0,len(outAlign[1])):
        listSweep.append(VmonAdc(outAlign[0],\
                                 outAlign[1][AP],\
                                 outAlign[2][AP],\
                                 outAlign[3][AP]))

    alignedArraySweep = np.asarray(listSweep).reshape((1,-1))

    return alignedArraySweep

def setIntervalAlignedArraySweep(arraySweep, alignedArraySweep, smoothingFactor):
    '''

    defining intervals in array of aligend data

    '''
    for column in range(0, arraySweep.shape[1]):
        listLow = []

        columnSweep = arraySweep[:,column]
        for AP in columnSweep:
            listLow.append(AP.Datapoints[0][0])

        APMinLow = listLow.index(min(listLow))
        APMaxLow = listLow.index(max(listLow))

        alignedArraySweep[0, column].setSmoothingFactor(smoothingFactor)
        alignedArraySweep[0, column].univariateSplineFit(-(len(arraySweep[APMaxLow][column].Xcoord)-int(np.argwhere(arraySweep[APMaxLow][column].Xcoord == arraySweep[APMaxLow][column].Interval[0][0]))),\
                                                         int(np.argwhere(arraySweep[APMinLow][column].Xcoord == arraySweep[APMinLow][column].Interval[0][-1])))


def getResultsArray(arraySweep):
    '''

    creating array just containing time(X)/current(Y)-Values of the peaks

    Parameters
    ----------
    arraySweep : array
        array of objects(VmonAdc), one train/trace per row

    Returns
    -------
    resultsArray : array
        array of tuples, each tuple contains time(X)/current(Y) values (Xmin, Ymin, Xmax, Ymax)

    '''
    resultsArray = np.empty((0, arraySweep.shape[1], 4),list)

    for row in range(0, arraySweep.shape[0]):
        listRow = []
        for column in range(0, arraySweep.shape[1]):
            listRow.append((arraySweep[row,column].Datapoints[0][0],\
                            arraySweep[row,column].Datapoints[0][1],\
                            arraySweep[row,column].Datapoints[1][0],\
                            arraySweep[row,column].Datapoints[1][1]))
        resultsArray = np.concatenate((resultsArray, [listRow]), axis=0)

    return resultsArray



#functions to extract data from array sweep
def writeToClipboard(output):
    '''copying data into clipboard
    '''
    process = subprocess.Popen(
            'pbcopy', env={'LANG': 'en_US.UTF-8'}, stdin=subprocess.PIPE)
    process.communicate(output.encode('utf-8'))



def clipboardExcelInwardPeak(resultsArray):
    '''
    copying the Inward-peaks from the results array into the clipboard
    as a string using a format suitable for Excel
    '''
    clipboardString = ''

    for row in range(0,resultsArray.shape[0]):
        for column in range(0,resultsArray.shape[1]):
            clipboardString += str(resultsArray[row, column, 1])+'\t'
        clipboardString += '\r'

    writeToClipboard(clipboardString)
    print ('Inward-Peaks copied to clipboard')


def clipboardExcelPeakInterval(resultsArray):
    '''
    copying the time intervals between the peaks into the clipboard
    as a string using a format suitable for Excel
    '''
    clipboardString = ''

    for row in range(0,resultsArray.shape[0]):
        for column in range(0,resultsArray.shape[1]):
            clipboardString += str(resultsArray[row, column, 2]-resultsArray[row, column, 0])+'\t'
        clipboardString += '\r'

    writeToClipboard(clipboardString)
    print ('Intervals between peaks copied to clipboard')





arraySweep = makeArraySweep(firstSeries-1, lastSeries-1, firstSweep-1, lastSweep-1)

print ('ArraySweep')
print ('Rows:',arraySweep.shape[0])
print ('Columns:',arraySweep.shape[1])
print (arraySweep[0][0].Xinterval)
```

```
Using matplotlib backend: MacOSX
ArraySweep
Rows: 4
Columns: 20
1e-06
```

In [6]:

```python
'''
Cell#2: Manual fitting-interval selection

After executing the cell, the first stimulus of first train will be plotted in an extra window.

fitting interval is selected by clicking into the figure: select the window with the trace as the active window and then clicking twice onto the trace for defining interval borders
    - fitted curve for the selected interval will be shown
    - to redefine the interval again click on the trace to set interval borders
    - selected interval is confirmed by return key

in case of multiple column arrays: -first window shows first stimulus of first train
                                   -after confirming first selected interval, last stimulus of first train is shown -> interval can be selected (this time just one click for the start point is needed, end point is defined by length of the first interval) and confirmed the same way
                                   -remaining intervals are set by linear interpolation between start- and endpoints of first and last interval
                                   -for visual control, the first train with fitted curves is shown (both for signal and noise intervalls)
'''

setSmoothingFactorArray(arraySweep, 0.0005)

plt.close('all')
fig = plt.figure(figsize=(16,8))

plotObjectsArray(arraySweep, 0, 1, 0, 1)

intervalStartEndFirst = []
def chooseFirst(event):
    intervalStartEndFirst.append(arraySweep[0][0].getDatapoint(event.xdata-arraySweep[0,0].Xcoord[0]))
    print ('-')

    if len(intervalStartEndFirst) == 2:
        arraySweep[0,0].univariateSplineFit(intervalStartEndFirst[0],intervalStartEndFirst[1])

        plotObjectsArray(arraySweep, 0, 1, 0, 1)

        print (intervalStartEndFirst[0]*arraySweep[0,0].Xinterval,intervalStartEndFirst[1]*arraySweep[0,0].Xinterval)
        intervalStartEndFirst.clear()
        acceptFirstInterval = fig.canvas.mpl_connect('key_press_event', acceptFirst)

def acceptFirst(event):
    if event.key == 'enter':
        print ('---------------')

        fig.canvas.mpl_disconnect(chooseFirstInterval)
        chooseLastInterval = fig.canvas.mpl_connect('button_press_event', chooseLast)
        acceptLastInterval = fig.canvas.mpl_connect('key_press_event', acceptLast)

        plotObjectsArray(arraySweep, 0, 1, arraySweep.shape[1]-1, arraySweep.shape[1])

intervalStartEndLast = []
def chooseLast(event):

    intervalStartEndLast.append(arraySweep[0][0].getDatapoint(event.xdata-arraySweep[0,arraySweep.shape[1]-1].Xcoord[0]))
    print ('-')

    startIntervalRange = [int(np.argwhere(arraySweep[0,0].Xcoord == arraySweep[0,0].Interval[0][0])),\
                          int(np.argwhere(arraySweep[0,0].Xcoord == arraySweep[0,0].Interval[0][-1]))]

    arraySweep[0,-1].univariateSplineFit(intervalStartEndLast[0],intervalStartEndLast[0]+startIntervalRange[1]-startIntervalRange[0])

    plotObjectsArray(arraySweep, 0, 1, arraySweep.shape[1]-1, arraySweep.shape[1])

    print (arraySweep[0,-1].Interval[0][0], arraySweep[0,-1].Interval[0][-1])
    intervalStartEndLast.clear()

def acceptLast(event):
    if event.key == 'enter':
        print ('---------------')

        startInterval = [int(np.argwhere(arraySweep[0,0].Xcoord == arraySweep[0,0].Interval[0][0])),\
                         int(np.argwhere(arraySweep[0,arraySweep.shape[1]-1].Xcoord == arraySweep[0,arraySweep.shape[1]-1].Interval[0][0]))]

        stepStartInterval = np.interp(range(1,arraySweep.shape[1]+1), [1,arraySweep.shape[1]], startInterval)

        endInterval = [int(np.argwhere(arraySweep[0,0].Xcoord == arraySweep[0,0].Interval[0][-1])),\
                       int(np.argwhere(arraySweep[0,arraySweep.shape[1]-1].Xcoord == arraySweep[0,arraySweep.shape[1]-1].Interval[0][-1]))]

        stepEndInterval = np.interp(range(1,arraySweep.shape[1]+1), [1,arraySweep.shape[1]], endInterval)

        for row in range(0, 1):
            for column in range(0, arraySweep.shape[1]):
                arraySweep[row, column].univariateSplineFit(int(stepStartInterval[column]),int(stepEndInterval[column]))

        plotObjectsArray(arraySweep, 0, 1, 0, arraySweep.shape[1])

chooseFirstInterval = fig.canvas.mpl_connect('button_press_event', chooseFirst)

if arraySweep.shape[1] == 1:
    fig.canvas.mpl_disconnect(chooseFirstInterval)
    chooseLastInterval = fig.canvas.mpl_connect('button_press_event', chooseLast)
    acceptLastInterval = fig.canvas.mpl_connect('key_press_event', acceptLast)
```

```
-
-
0.0032069999999999998 0.004047
---------------
-
0.291358 0.292198
---------------
---------------
```

Cell_2

In [7]:

```python
'''
Cell#3:
Noise is analysed similarly to enable discrimination of current peak amplitudes from noise amplitudes.
Noise levels preceding individual current peaks are fitted similarly to current peeks and provided as noisePeak

- the gap between the end of the noise interval and the start of the peak interval (defined in cell #2) is provided
- NaPeak and NaNoise interval for fitting have the exact same lenght
- fittedNoise curves are plotted similiar to peak fitting above
'''

plt.close('all')
fig = plt.figure(figsize=(16,8))

startIntervalRangeTime = arraySweep[0,0].Interval[0][-1]-arraySweep[0,0].Interval[0][0]
gapNoise = 0.001      #gap in s between end of Noise fitting interval and in cell #2 defined startpoint


#creating array with Noise objects
NoiseArray = makeArraySweep(firstSeries-1, lastSeries-1, firstSweep-1, lastSweep-1,\
                            startTime = startTime-startIntervalRangeTime-gapNoise)

print ('Noise')
print ('Rows:',NoiseArray.shape[0])
print ('Columns:',NoiseArray.shape[1])

setSmoothingFactorArray(NoiseArray, 0.0003)



startInterval = [int(np.argwhere(arraySweep[0,0].Xcoord == arraySweep[0,0].Interval[0][0])),\
                 int(np.argwhere(arraySweep[0,-1].Xcoord == arraySweep[0,-1].Interval[0][0]))]

stepStartInterval = np.interp(range(1,arraySweep.shape[1]+1), [1,arraySweep.shape[1]], startInterval)

endInterval = [int(np.argwhere(arraySweep[0,0].Xcoord == arraySweep[0,0].Interval[0][-1])),\
               int(np.argwhere(arraySweep[0,-1].Xcoord == arraySweep[0,-1].Interval[0][-1]))]

stepEndInterval = np.interp(range(1,arraySweep.shape[1]+1), [1,arraySweep.shape[1]], endInterval)


#fitting remaining trains from arraySweep, in cell #2 just the first train was fitted
for row in range(1, arraySweep.shape[0]):
    for column in range(0, arraySweep.shape[1]):
        arraySweep[row, column].univariateSplineFit(int(stepStartInterval[column]),int(stepEndInterval[column]))

#fitting Noise in given Intervals
for row in range(0, NoiseArray.shape[0]):
    for column in range(0, NoiseArray.shape[1]):
        NoiseArray[row, column].univariateSplineFit(int(stepStartInterval[column]),int(stepEndInterval[column]))

#resetting baseline according to first points in each stimulus, thereby compensating drifts during the train
setBaselineCorrectionBeforeIntervalsArray(NoiseArray, 0.002, 0.001) #in s distance to start of interval

plotObjectsArray(NoiseArray, 0, 1, 0, NoiseArray.shape[1])
```

```
Noise
Rows: 4
Columns: 20
```

Cell_3

In [8]:

```python
'''
Cell#4 resetting baseline by using setBaselineCorrectionBeforeIntervalsArray(), for further information see functions in cell#1
'''

plt.close('all')
fig = plt.figure(figsize=(16,8))

setBaselineCorrectionBeforeIntervalsArray(arraySweep, 0.002, 0.001)
plotObjectsArray(arraySweep, 0, 1, 0, arraySweep.shape[1])
```

Cell_4

In [9]:
```python
'''
Cell#5 calculating averaged train (stimuli aligned by Inward-Peak), for further information see functions in cell#1
'''

plt.close('all')
fig = plt.figure(figsize=(16,8))

outAlign = alignLow(arraySweep)
alignedArraySweep = makeAlignedArraySweep(outAlign)

print ('AlignedArray')
print ('Rows:',alignedArraySweep.shape[0])
print ('Columns:',alignedArraySweep.shape[1])

setIntervalAlignedArraySweep(arraySweep, alignedArraySweep, 0.0006)
plotObjectsArray(alignedArraySweep, 0, 1, 0, alignedArraySweep.shape[1])
```

```
AlignedArray
Rows: 1
Columns: 20
```

Cell_5

In [10]:
```python
'''
Cell#6 getting results, for further information see functions in cell#1
'''

#setting Inward-Peak to t=0 before getting the results (Inward-Peak-/Outward-Peak-amplitudes, time between peaks)
setLowZeroArray(arraySweep)
setLowZeroArray(alignedArraySweep)
setLowZeroArray(NoiseArray)

#creating array of results (same structure as arraySweep)
resultsArray = getResultsArray(arraySweep)
resultsNoise = getResultsArray(NoiseArray)
resultsAlignedArray = getResultsArray(alignedArraySweep)

#copying results to clipboard
clipboardExcelInwardPeak(resultsArray)
#clipboardExcelPeakInterval(resultsArray)

#plotObjectsArray(arraySweep, 0, arraySweep.shape[0], 0, 1)
```

```
Inward-Peaks copied to clipboard
```

In [11]:
```python
'''
Cell#7 creating .txt-files for further analysis and plotting in Igor

Each .txt-file contains a table, columns are seperated by tabulator

The number in each filename refers to the number of the stimulus in the train.
Analyzing trains containing 20 stimuli will result in 20 txt-files.

Unaveraged-files, contain rare data (each stimulus of the train)
Averaged-files, contain averaged data

The data (actual current values, fitted current values, peak-values) is furthermore split into three categories:
Data[...],Fit[...] and Peak[...]

Data[...]-files contain the time values in the first column and the current value of each train in the follwoing columns.
Example: If 4 trains are analyzed the arraySweep contains 4 rows. The Data[...]-file is going to comtain 5 columns, one for time values and 4 more for the values from the trains.

Fit[...]-files contain the fitted time and current values.
Example: If 4 trains are analyzed the arraySweep contains 4 rows. The Fit[...]-fule is going to contain 8 columns.

Peak[...]-files contain the time and current values of both peaks. Each row (made out of 4 columns) contains the time/current value for the Inward-peak and the time/current value for the Outward-peak.
'''

import os, re, os.path

#creating Data folder, if not existent
if not os.path.exists('Data'):
    os.makedirs('Data')

#deleting all files in data folder
for root, dirs, files in os.walk('Data'):
    for file in files:
        os.remove(os.path.join(root, file))

def makeTxtFilesArray(arraySweep, name):
    for column in range(0, arraySweep.shape[1]):
        listLowIndex = []

        columnSweep = arraySweep[:,column]
        for AP in columnSweep:
            listLowIndex.append(int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0])))
            #print (int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0])))
            #print (AP.Acoord[int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0]))])

        IndexMinLow = min(listLowIndex)
        IndexMaxLow = max(listLowIndex)

        DataTxtArray = []
        FitTxtArray = []
        PeaksTxtArray = []

        for AP in columnSweep:
            startWindow = int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0]))-IndexMinLow
            endWindow = int(np.argwhere(AP.Xcoord == AP.Datapoints[0][0]))+(len(AP.Xcoord)-1-IndexMaxLow)

            DataTxtArray.append(AP.Acoord[startWindow:\
                                          endWindow+1])

            FitTxtArray.append(AP.Interval[0,:])
            FitTxtArray.append(AP.Interval[1,:])

            PeaksTxtArray.append([AP.Datapoints[0][0],AP.Datapoints[0][1],AP.Datapoints[1][0],AP.Datapoints[1][1]])


        Xcoords = []
        for coord in range(0,len(DataTxtArray[0])):
            Xcoords.append((coord-IndexMinLow)*arraySweep[0][column].Xinterval)

        DataTxtArray.insert(0, Xcoords)

        DataTxtArray = np.asarray(DataTxtArray)
        DataTxtArray = DataTxtArray.transpose()

        FitTxtArray = np.asarray(FitTxtArray)
        FitTxtArray = FitTxtArray.transpose()

        PeaksTxtArray = np.asarray(PeaksTxtArray)

        #print (column+1)
        #print (int(np.argwhere(DataTxTArray[:,0] == 0)))
        #for column in range (1, DataTxtArray.shape[1]):
        #    print (int(np.argwhere(DataTxTArray[:,column] == )))

        np.savetxt('Data/Data%s_%d.txt'%(name, column+1), DataTxtArray, delimiter='\t')
        np.savetxt('Data/Fit%s_%d.txt'%(name, column+1), FitTxtArray, delimiter='\t')
        np.savetxt('Data/Peaks%s_%d.txt'%(name, column+1), PeaksTxtArray, delimiter='\t')

makeTxtFilesArray(arraySweep, 'UnaveragedAP')
makeTxtFilesArray(alignedArraySweep, 'AveragedAP')
```