Analysis of cardiac reversal

Heart videos were recorded at 30 fps. Then datasets from the tracking of the nuclei were first regularized by subtracting mean and dividing by standard deviation. Then data were resampled to 60 fps by interpolation of data points. That means that we calculated a value between each two data points creating a signal at 60 fps.

Then, we first identify peaks and troughs in the displacement of both nuclei. For a cell to be considered as beating first its peak/trough had to be 1 to 7 frames ahead of the peak/trough of the second cell. When the anterior cell peaked/troughed first at least 4 times in a row we classified it as a bout in backward direction. Conversely, when the posterior cell peaked/troughed first at least 4 times in a row we classified it as a bout in forward direction. Simultaneous peaks/troughs correspond to those where both cardiomyocytes showed maximal displacement in the same frame. These were infrequent, corresponding on average to 10% of the recorded peaks/troughs, and always isolated.

Having all anterograde, retrograde and simultaneous peaks/troughs identified, we next defined bouts of pumping either in the backward or forward direction, and points of reversal. For a sequence of beats in a particular direction to be considered a bout, at least 2 complete consecutive beats (a sequence of 2 peaks and two troughs) in the same direction were required. If a long sequence of beats in one direction, for example a forward bout, was interrupted by < 2 consecutive beats (at most one peak and 2 troughs or two peaks and one trough) in the opposite direction, we considered the whole sequence as an uninterrupted bout, in this example a single forward bout. If, however, the interruption consisted of more than 2 beats (a sequence containing 2 peaks and 2 troughs) in the same direction, then we considered the pumping direction changed counting as a cardiac reversal. To determine the timepoints of cardiac reversal, we relied on the beginning and end of the backward pumping bouts as cardiac beating in these were more regular than in the forward bouts.

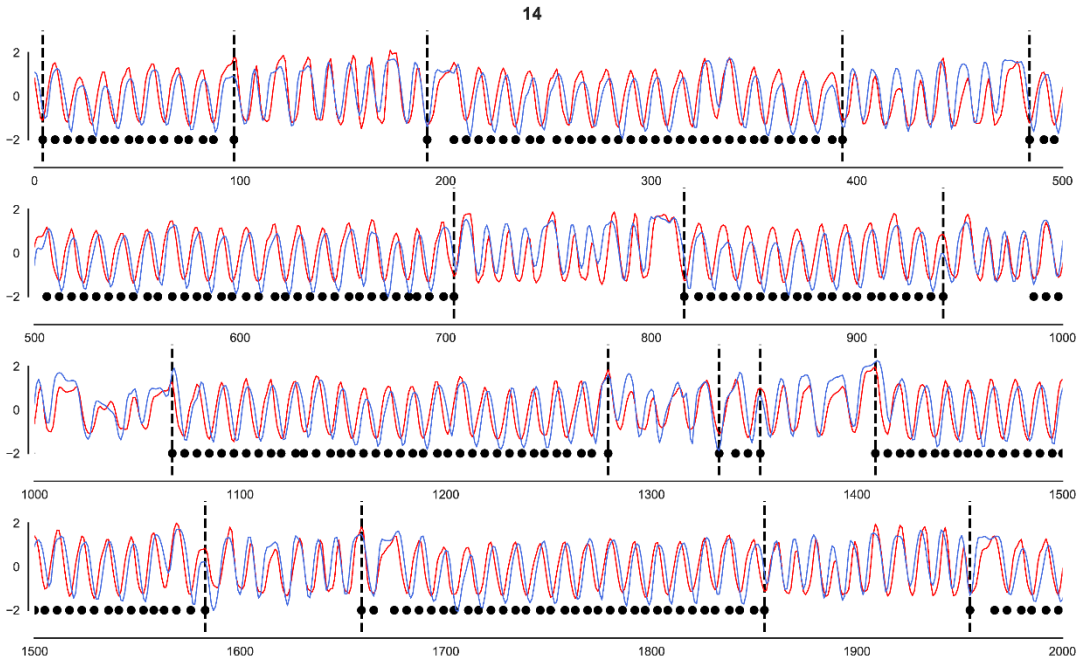
Selected examples of raw traces

In each figure the displacement of the anterior and posterior nuclei is showed in red and blue respectively. Vertical dashed lines represent identified points of cardiac reversal. Black dots indicate identified backward peaks/troughs.

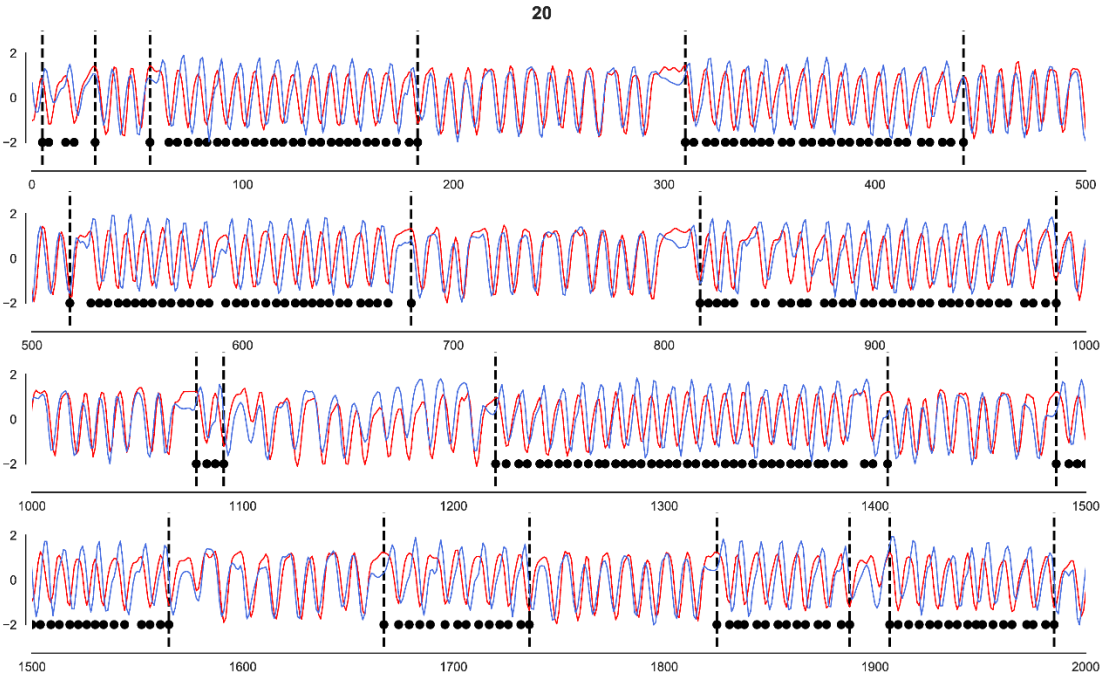
For each animal we have plotted the first 2000 frames (33,3 sec).

1) Examples of traces of animals included

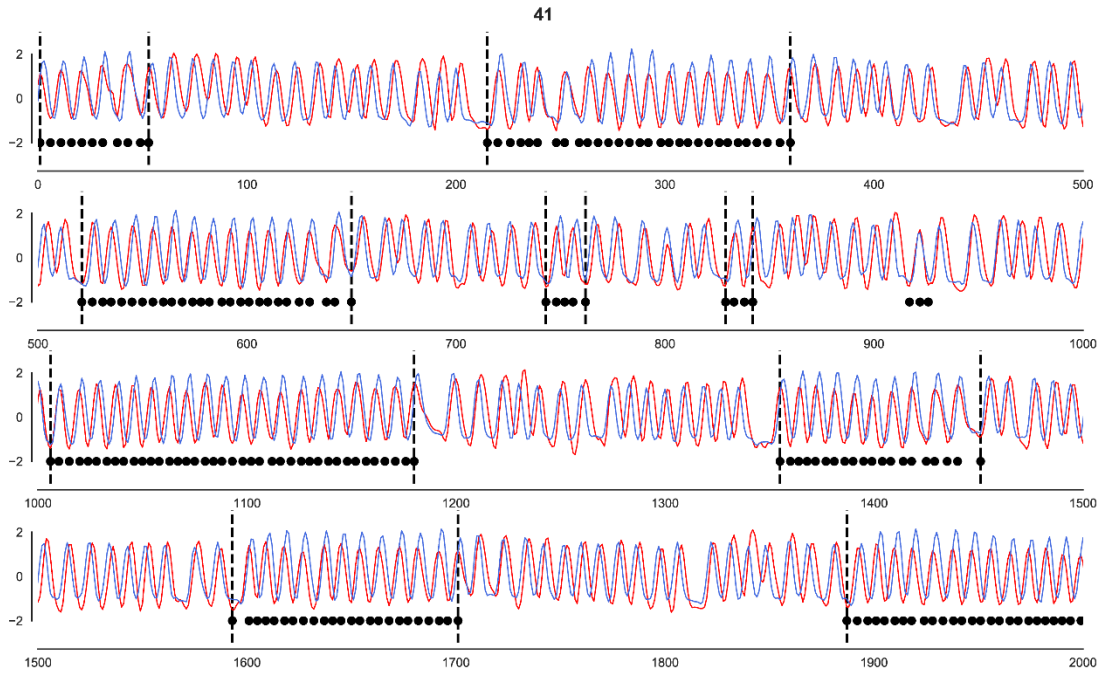
Animal 1_14



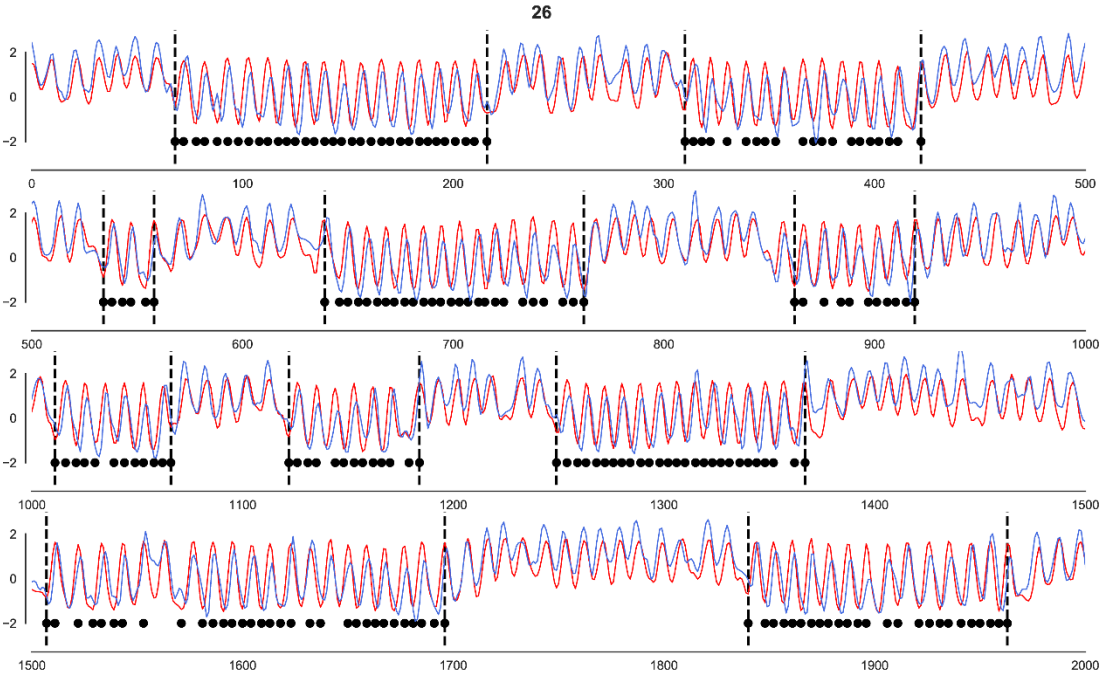
Animal 1_20



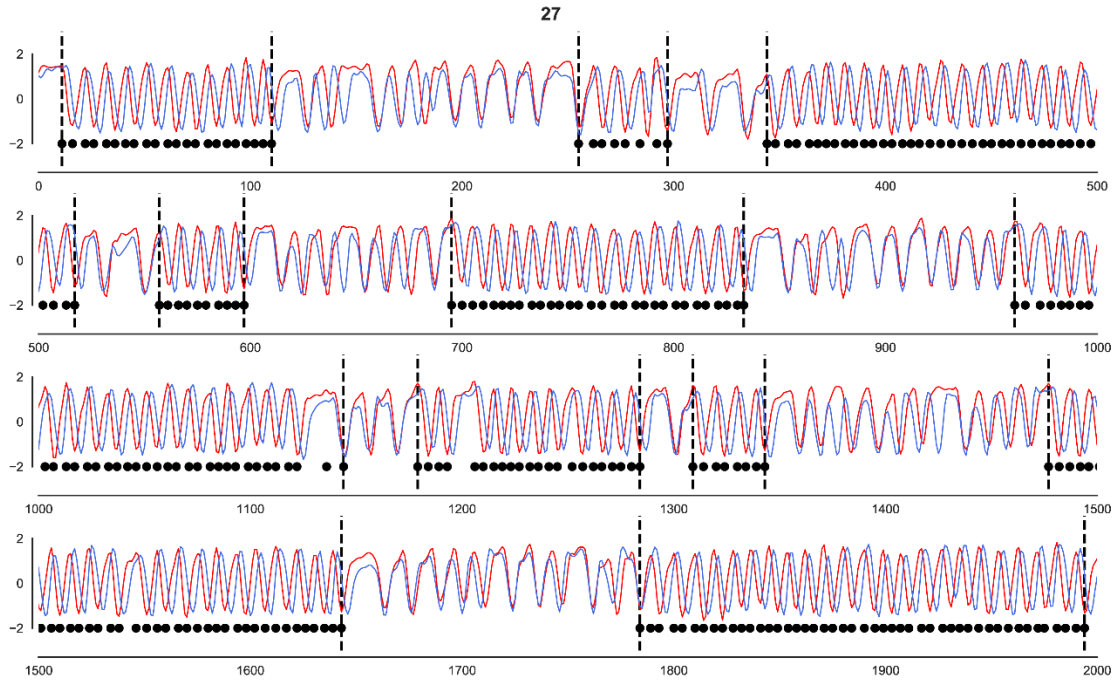
Animal 1_41



Animal 2_26



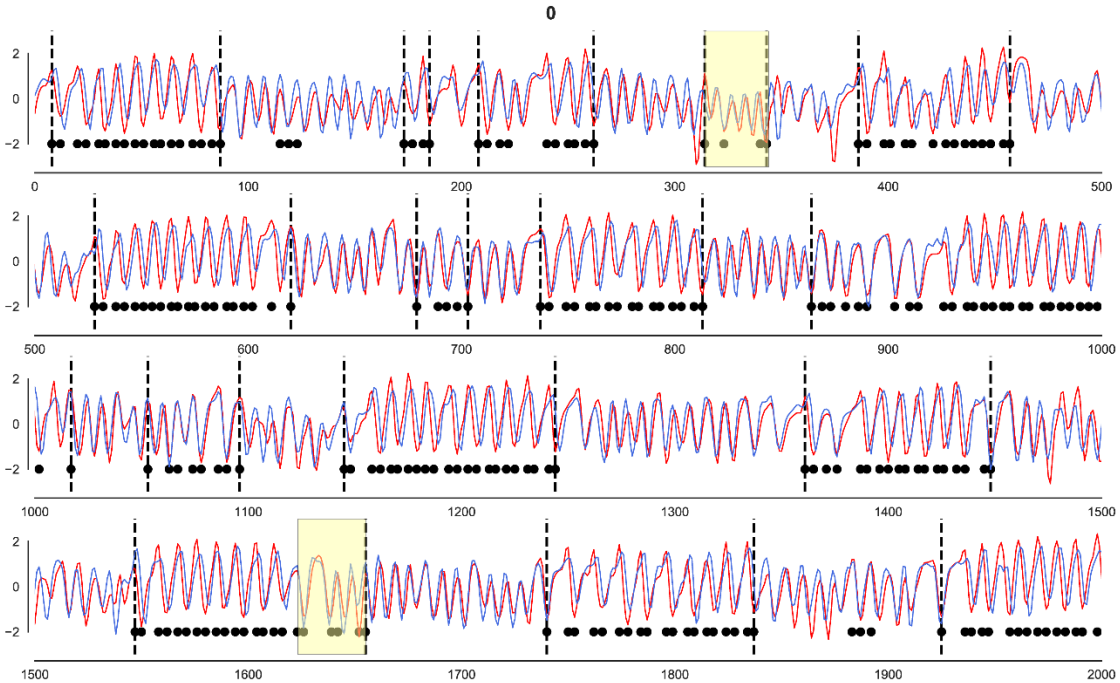
Animal 2_27



2) Example of errors in traces of included animals.

Animal 1_0

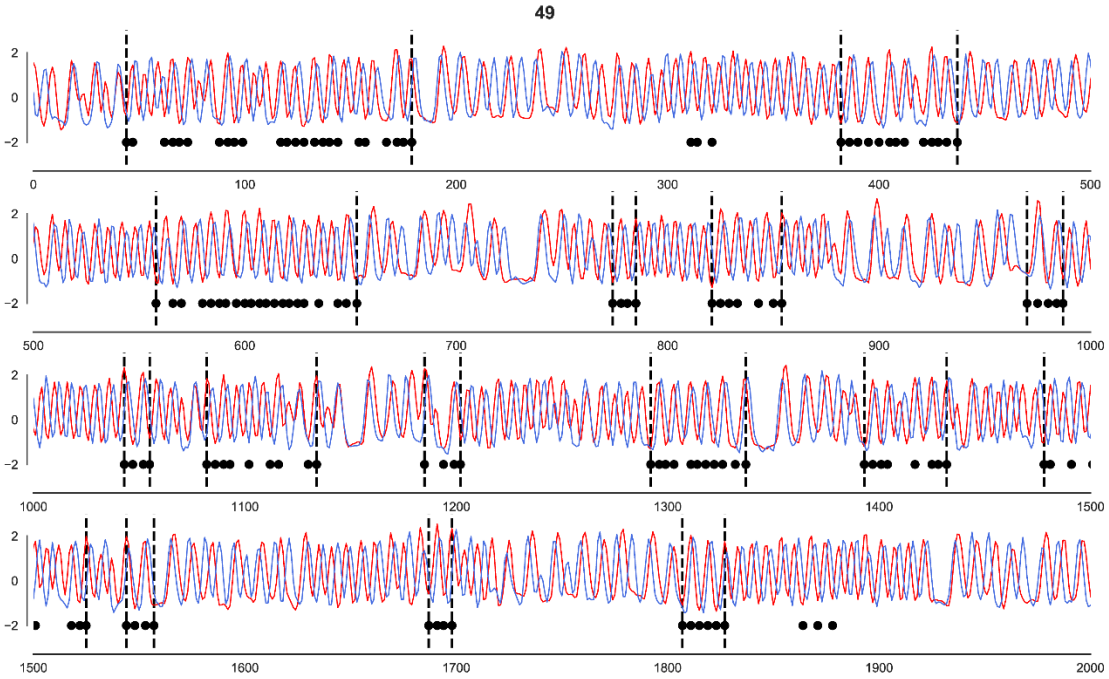
In this case, a small fraction of forward beatings are included as backward pumping (yellow).



3) Examples of traces of other excluded animals.

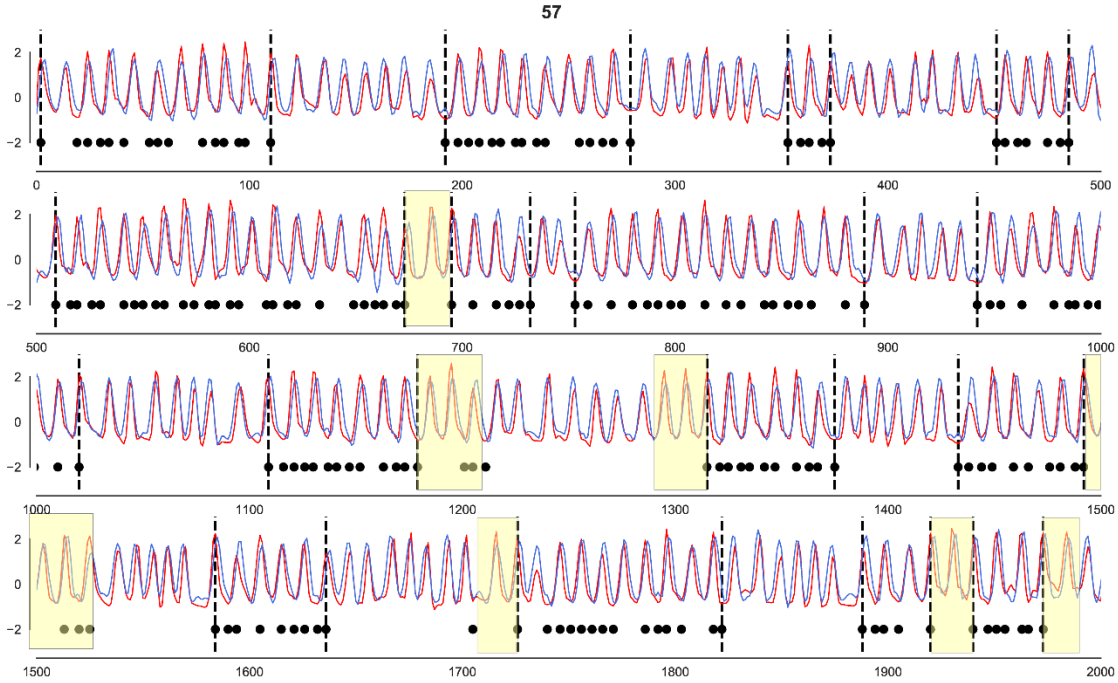
Animal 2_49

This animal was excluded because anterior and posterior peaks/troughs are too separated and it was not possible to automatically diferentiate backward from forward pumping bouts.



Animal 2_57

Animal excluded because a fraction of backward beatings are included as forward pumping (yellow).



Code used to analyze cardiac reversal.

 # Function for troughs

```
def peaks_less(waveA,waveP, A=36078,P=36078):
    "Identify the peaks when waves are in same phase"
    "And separate them in two list:indexes for retrograde beating and anterograde beatings "
    peaksmax_A = np.array(argrelextrema(waveA, np.less_equal,order=6))
    peaksmax_P = np.array(argrelextrema(waveP, np.less_equal,order=6))

    peaksM_A= peaksmax_A[0]
    peaksM_P= peaksmax_P[0]
    PeaksAList = [0] * A
    for i in peaksM_A:   #add the index of peaks
        PeaksAList[i] = i
    PeaksPList = [0] * P
    for j in peaksM_P:       #add the index of peaks
        PeaksPList[j] = j
    Listpeaks = []    #Creates a list where in each frame says if peaks happens in anterior 'ant'  posterior 'post or both 'anpo'
    for i in range(0, len(PeaksPList)):
        if PeaksAList[i] == 0 == PeaksPList[i]:
            Listpeaks.append(None)
        else:
            if PeaksAList[i] > PeaksPList[i]:
                Listpeaks.append('ant')
```

```
        else:
            if PeaksAList[i] == PeaksPList[i]:
                Listpeaks.append('anpo')
            else:
                if PeaksAList[i] < PeaksPList[i]:
                    Listpeaks.append('post')


    list_backward= []  #list of index of peaks coresponding only to retrograde beating (anterior cell beating first)
    list_foreward= []   #list of index of peaks coresponding only to anterograde beating (posterior cell beating first)
    for i in range(0, len(Listpeaks)-12):
        if Listpeaks[i] == None:
            pass
        else:
            if Listpeaks[i] == 'anpo':# and Listpeaks[i-1] == None and Listpeaks[i-2] == None and Listpeaks[i-3] == None and Listpeaks[i-4] == None:
                list_backward.append(i)
            #pass
            elif Listpeaks[i] == 'post' and Listpeaks[i-1]== None and Listpeaks[i-2]== None and Listpeaks[i-3]== None and Listpeaks[i-4]== None:
                for b in range(1,7):
                    if Listpeaks[i+b] == 'ant':
                        list_foreward.append(i)
                    else:
                        pass
```

```
        elif Listpeaks[i] == 'ant' and Listpeaks[i-1]== None and Listpeaks[i-2]== None and Listpeaks[i-3]== None and Listpeaks[i-4]== None:
            for b in range(1,7):
                if Listpeaks[i+b] == 'post':
                    list_backward.append(i)
                else:
                    pass
    return list_backward, list_foreward
```

#Function for peaks

```
def peaks_greater(waveA,waveP, A=36078,P=36078):
    "Identify the peaks when waves are in same phase"
    "And separate them in two list:indexes for retrograde beating and anterograde beatings "

    peaksmax_A = np.array(argrelextrema(waveA, np.greater_equal,order=6))
    peaksmax_P = np.array(argrelextrema(waveP, np.greater_equal,order=6))
```

 # Separate in different arrays each backward pulse and each forward pulse

```
def direction(list_1, list_2):
    "Separate in different arrays each backward pulse and each forward pulse"
    "these lists only contain indexes for the peaks"
    portions= np.split(list_1,np.searchsorted(list_1, list_2))
    return portions
```

 # Run the function peaks for each animal and got peaks according to direction in anterior cell

```
backward_pulses_t=[]
for ani in range(0,len(Heart_total),2):  # Identify high and lower peaks and add them
    back1, fore1= peaks_greater(Heart_total[ani],Heart_total[ani+1],A=36078,P=36078)
    back2, fore2= peaks_less(Heart_total[ani],Heart_total[ani+1],A=36078,P=36078)
    fore3=fore1+fore2
    back3=back1+back2
    w= sorted(fore3)
    y= sorted(back3)

    backward=direction(y,w)
    back_clean=[]
    for each in backward:
        if len(each)>2:
            for e in each:
                back_clean.append(e)
    backward_pulses_t.append(back_clean)
```

 # Remove those peaks that are too close from the previous one

```
for ani in backward_pulses_t:
    bck_deleted=[]
    for i in range(0,1):
        if ani[i+1]< ani[i]+3:
            bck_deleted.append(i)
```

```
    for i in range(1, len(ani)-1):
        if ani[i+1]< ani[i]+3:# or ani[i-1]> ani[i]-3:
            bck_deleted.append(i)
    for i in range(len(ani)-1, len(ani)):
        if ani[i]< ani[i-1]+3:
            bck_deleted.append(i)
    for index in sorted(bck_deleted, reverse=True):
        del ani[index]
```

 # Slipt the peaks in fragments according to the distance among them

```
t_periods_backward=[]
for ani in backward_pulses_t:
    back_frag=[]
    array=[]
    array1=[]
    for i in range(0, len(ani)-1):
        if ani[i]+18 >= ani[i+1]:
            array.append(ani[i])
        else:
            array.append(ani[i])
            back_frag.append(array)
            array=[]
    back_per=[]
    for x in back_frag:            # Remove those fragments with less tham 3 peaks.
```

```
    if len(x)>3:
        back_per.append(x)
t_periods_backward.append(back_per)
```