Labjournal zebrafish

February 7, 2017

1 Lab journal - Zebrafish data

Casper van Elteren, 2016

1.1 Aims:

- Data exploration of the zebrafish
- Working on big data
- Build from analysis scripts in python 3.5
- Work with Jupyter notebook
- Work with hdf5 files

1.2 Current working on :

- Dimensionality reduction
- zebrafish viewer

1.3 Introduction

A fundamental question in neuroscience is how neural networks generate behavior and adapt to changes in sensory feedback. In 2012 Ahrens et al. used two-photon calcium imaging to record the activity of large populations of neurons at the cellular level throughout the brain of the zebrafish. They were able to decompose the adaptive location of the fish into four types of neuronal response, and provide anatomical maps of the corresponding neural networks underlying this behavior. In this labreport we have obtained data from the Ahrens lab containing over 90 000 cells and ~ 1 hour long recording of a spontaneously behaving fish. Initially, the idea was to look for temporal structure by means of reverse correlation, however due to the very high computation time this approach was abandoned. Focus was shifted to provide work on data viewer for the fish, to use for later use in master thesis that would analyze this dataset more thoroughly.The code was built from the ground up in Python 3.5, the time of the lab report was mostly put in developing the code. This labreport was used as an initial step in the master thesis that would data-mine the data from several zebrafish provided by the Ahrens lab, and is in collaboration with Marcel van Gerven's lab.

1.4 LOGs

As is mentioned in the introduction, the initial approach was to look for temporal structure using reverse correlation between all cells. It was known by previous studies from the Ahrens lab that

there is a distinct hindbrain oscillator that is implied in producing motion. The idea was to focus more on the forebrain which showed conflicting results in different studies. By first trying to replicate the results from Ahrens et al. (2012), we would validate our approach and focus on tackling the forebrain. However, this approach was abandoned due to the computation time. The log entries below show the process of working on the reverse correlation. Afterwards, the focus was shifted to provide a data viewer for the data, i.e. an app that could scroll through time and z-axis in order to make sense of the data.

6 / 11 : - Started computing the cross-correlation over the cells - First used the Ahrens script to compute the cells that show a high correlation with the 'Frame Turn' vector - About 8000 cells showed high correlation, I used these cells for the cross-correlation - Due to the limitation of storage, I'm only writing to disk cells that show a maximum correlation over time abs(r) > .5 9 / 11 : - Running of the analysis took way too long with standard np.correlate and statsmodels.tsa.ccf (very slow). - Furthermore, statsmodel gives artefacts at long time lags

10 / 11 : - Implemented my own cross-correlation based on fourier transform; should be fast than convolution anyways - Normalizing the data over number of time-points gives correlation between -1,1
12 / 11 : - Finally have the results

14 / 11 : - Aim is to visualize the data in 3d to look for possible clusters in the data - It works - Exporting to mp4 -> works - Back of the brain seems well connected! 15 / 11 : - How to analyze these data? Can we see a pattern? - Cell 1267 is a frontally located cell that shows promise of correlating welll with the back of the brain 17 / 11 : - Clean up code; make template brain, extract indices of front brain cells - Re-run autocorrelation on frontal brain + cross 20 - 11 : - Implemented a function that cross-checked the cross-correlation with the statsmodel implementation (and matlab implementation; my implementation is faster, sum squared errors produces reasonable results, not sure what ground truth is though 22 - 11 : - Zebra-fish viewer initial startup using matplotlib 24 - 11: - Continue working on fish viewer (fv) 25 - 11: - forked fv to use with classes 26 - 11: - Autocorrelation speed up using parallel programmming : in progress 27 - 11: - bug fixing; need a faster way to access the cross-correlation; looked into ASSET, but only used for spiking data

28- 11: - running cross-correlation for all front brain cells (takes long time)

1.5 Outstanding isssues

- use fft with power of 2 for vast performance?
- ~~Need a 2D model brain to project data to –> backdrop of cell indices
- ~~Not taking into account the cells that are not well correlated with the movement, but are are well correlated with other cells~~ –> forebrain
- Using the swim-bouts as epochs to infer relevant structures
- There are some 'cell' that do not seem to fit the profile of the brain from the zebrafish atlas > remove?

1.6 Important notes

Keep in mind how indexing works; - Select the cells you want - Load the cells in cross-correlation - Cells in the saved file are relative to the size of the selected cells, - But are linear in the selection vector

2 Data visualisation

The following will contain some data visualization of the zebrafish data. The zebrafish viewer needs to be called as a separate module, please see the code.

2.1 Helper functions

Pre-amble loading necessary packages for plotting, and/or analysis

```
In [1]: # PRE-AMBLE
        import matplotlib
        matplotlib.use('TKAgg')

        from pylab import *
        from mpl_toolkits.mplot3d import Axes3D
        import matplotlib.animation as manimation

        import numpy as np

        import seaborn as sb

        import scipy
        import scipy.signal, scipy.io
        import sklearn


        from h5py import File

        file = '../Data/data.hdf5'
```

3 File structure

```
In [16]: with File(file,'r') as f:
             print('Data:\n')
             for i in f: print(i)
```

```
Data:

autocorrelation
coordinates
cross correlation
fft
filtered
filtered fft
front cells
normalized
raw
template
```

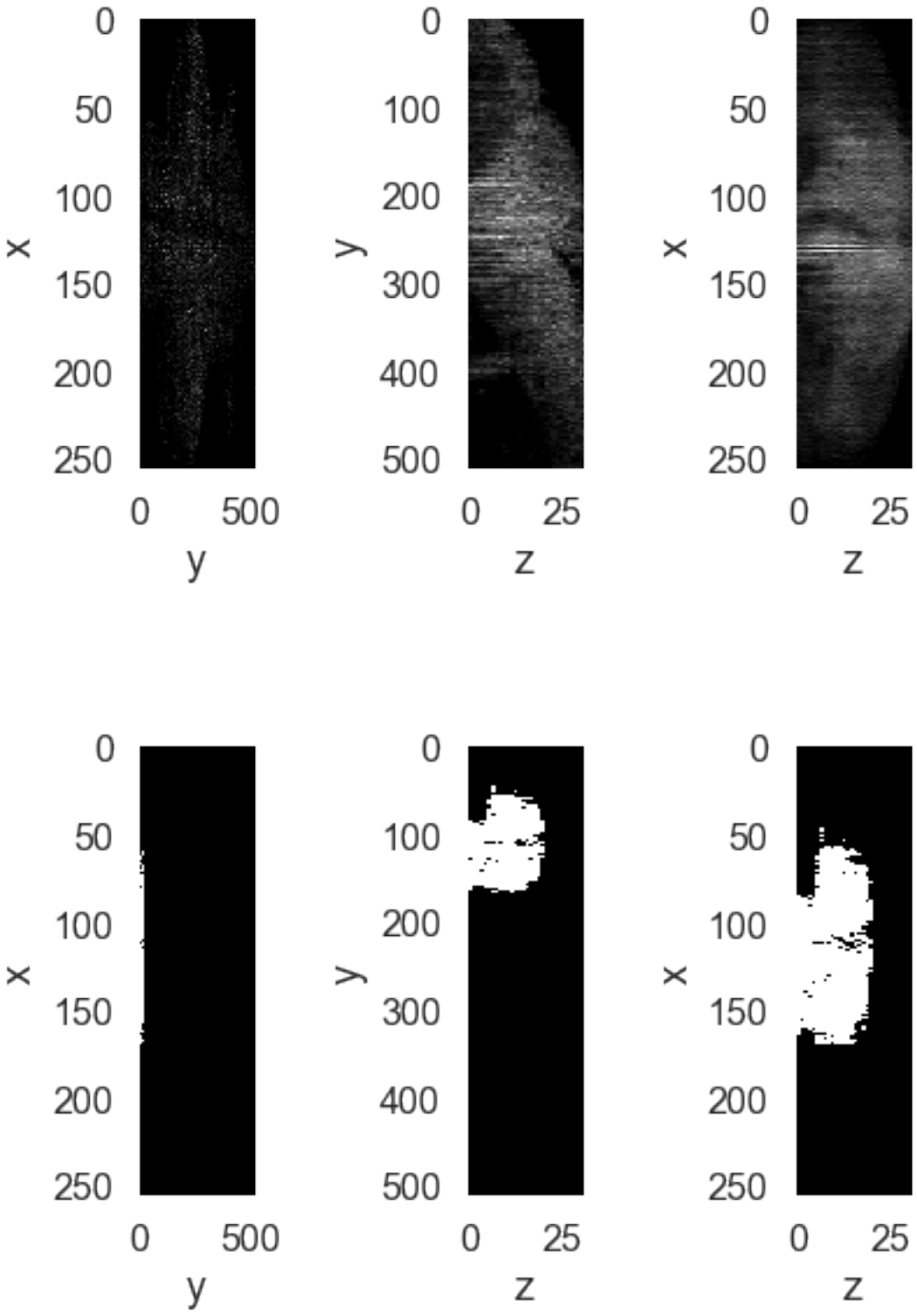
3.1 Zebrafish templates

This code will plot flattened maps at different angles for the cellbodies extracted in the data file.

```
In [17]: # Index gotten from pic above
         with File(file) as f:
             template = f['template'] # mask of where cells are
         #     for i in f['front cells']: print(i)
             for i in template: print(i)

             # load the coordinates of cells
             coord = \
             np.array([ f['coordinates'][i].value for i in f['coordinates']]).squee

             # front cell indices
             fc_idx   = f['front cells']['names'].value
             views    = ['xy', 'yz', 'xz']


             # plot the different views:
             # x-y, y-z, x-z
             %matplotlib inline
             fig1, ax1 = subplots(1, len(views))
             fig2, ax2 = subplots(1, len(views))

             for idx, view in enumerate(views):
                 data = template[view].value
                 if i == 0:
                     ax1[idx].imshow(data.T,\
                                     cmap ='gray',\
                                     origin = 'upper')
                     ax1[idx].set_xlabel(view[0])
                     ax1[idx].set_ylabel(view[1])

                     ax2[idx].set_xlabel(view[0])
                     ax2[idx].set_ylabel(view[1])
                 else:
                     ax1[idx].imshow(data,\
                                     aspect = 'auto',\
                                     cmap ='gray')
                     ax1[idx].set_xlabel(view[1])
                     ax1[idx].set_ylabel(view[0])

                     ax2[idx].set_xlabel(view[1])
                     ax2[idx].set_ylabel(view[0])

                 # empty container
                 data.fill(0)
                 if i == 0:
```

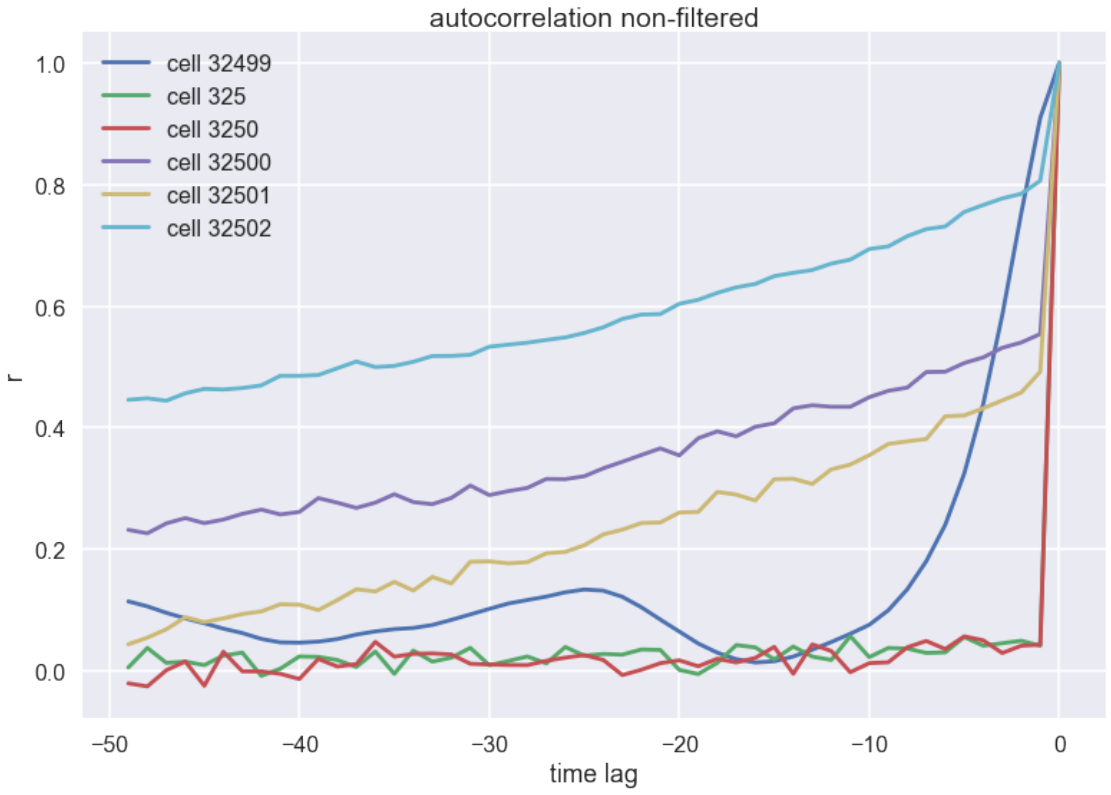
```
                    data[coord[0, fc_idx], coord[1, fc_idx]] = 1
                    ax2[idx].imshow(data.T,\
                                aspect = 'auto',\
                                cmap = 'gray')
                elif i == 1:
                    data[coord[1, fc_idx], coord[1, fc_idx]] = 1
                    ax2[idx].imshow(data,\
                                aspect = 'auto',\
                                cmap = 'gray')
                else:
                     data[coord[0, fc_idx], coord[2, fc_idx]] = 1

                ax2[idx].imshow(data,\
                                aspect = 'auto',\
                                cmap = 'gray')

                ax1[idx].grid('off')
                ax2[idx].grid('off')
            sb.set_context('poster')

            fig1.tight_layout()
            fig2.tight_layout()
            print('number of front cells = ', len(find(fc_idx == 1)))
            show()
```

```
xy
xyz
xz
yz
number of front cells =  8434
```



As can be seen in the figure, whole brain recordings are possible with this model organism. In order to study the forebrain, as mask was applied which is shown in the bottom row.

```
In [18]: n_range = 25000
         with File(file) as f:
             # obtain autocorrelation data [see provided code]
             gr = f['autocorrelation']
             c = 0
             for idx, i in enumerate(gr):
                 if idx > n_range:
                     plot(range(0,-50,-1), np.real(gr[i][:50]), label = 'cell {0}'.
         #             print(i)
                     c += 1
                 if c > 5:
                     break
         xlabel('time lag')
         ylabel('r')
         legend(loc = 0)
         title('autocorrelation non-filtered')
         show()
```



This figure shows the autocorrelation over time. We notice that some of the cells show prolonged activation over time, whereas others follow the drop-off expected as GCAMP2 was used.

```
In [19]: with File(file) as f:
             gr = f['normalized']
```

```
        c = 0
        for idx, i in enumerate(gr):
            if idx > n_range:

                plot(gr[i], label = 'cell {0}'.format(i))
                xlim([0, 100])
#                 print(i)
                c += 1
                if c > 5:
                    break

xlabel('time')
title('standardized $\Delta$ F/ F')
legend(loc = 0, ncol = 5)
show()
```

This figure shows the standardized $\Delta F/F$ for a subset of cell located in the forebrain.

```
In [20]: with File(file) as f:
             gr = f['filtered']
             c = 0
             for idx, i in enumerate(gr):
                 if idx > n_range:
```

```
                plot(gr[i], label = 'cell {0}'.format(i))
                xlim([0, 100])
                c += 1
                if c > 5:
                    break
    legend(loc = 0, ncol = 5)
    xlabel('time')
    title('filtered $\Delta$ F / F')
    show()
```

4 Fishviewer code

```
In [ ]: from pylab import *
        from matplotlib.widgets import *
        import numpy as np
        from scipy.io import loadmat
        from scipy.stats import zscore
        from h5py import File
        #%%
        close('all')
        class fish_viewer(object):
```

```
    def __init__(self,  file = 'tmp.hdf5'):
        self.file = file
        self.version                = '0.1'
        # initial values for plotting
        self.t_plot                 = 0
        self.z_plot                 = 0

        print('Welcome to the zebrafish viewer')
        print('Version = {0}'.format(self.version))
        print('loading data')


        with File(self.file) as f:
            # print(f.visit(print))
            cord = []
            data = []
            group = f['coordinates']
            for member in group:
                cord.append(group[member].value)
            # for i in f[name_group]: print(i); assert 0
            cord = np.array(cord).squeeze()
            # for idx, i in enumerate(f[name_group]):
            #      print(i)
            #      if idx > 100:
            #          break
            # # assert 0
            # self.namez = 1103
            # group = f[name_group][str(self.namez)]
            # for member in group:
            #     data.append(group[member].value)

            # # for i in group:print(i);
            # # assert 0
            group = f['front cells']['indices']
            tmp = []
            for member in group:
                tmp.append(int(member))
            self.fc_idx = np.array(tmp).flatten()
        print(self.fc_idx.shape)
#        print(len(tmp))
        # self.data = np.array(data)
        # print(self.data.shape)


        print('loading done')
        # self.data                   = data
        x, y, z                     = cord
```

```
        self.dims                   = np.max(cord, 1)
        self.x                      = x.flatten()
        self.y                      = y.flatten()
        self.z                      = z.flatten()
#        print(np.max(self.z[tmp]))
        # initialize plot matrix
        self.template               = \
            np.zeros((np.max(self.x) + 1, np.max(self.y) + 1))
#        print(np.max(x), np.max(y))


    def change_z_slider(self, val):
        self.z_plot = int(val)
        self.im.set_data(self.set_grid())
#        print(self.z_plot)
        self.z_slider.valtext.set_text('{0}'.format(self.z_plot))
        self.fig.canvas.draw()

    def change_time_slider(self, val):
        self.t_plot = int(val)
#        print(val)
        self.im.set_data(self.set_grid())
        self.time_slider.valtext.set_text('{0}'.format(-self.t_plot))
        self.fig.canvas.draw()

    def change_cell_slider(self, val):

        with File(self.file) as f:
            self.namez = self.fc_idx[int(val)]
            print('starting loading')
#            print(self.namez)
#            for i in f:print (i)
            gr = f['cross correlation'][str(self.namez)]

            tmp = []
            dd  = []
            for i in gr:
#                print(i)
                dd.append(int(i))
                tmp.append(gr[i].value)
        self.fc_idx = np.array(dd)
        self.data  = np.array(tmp)
#        print(">", self.data.shape)
        self.p.set_data(self.x[self.namez], self.y[self.namez])#, marker =
        self.time_slider.valtext.set_text('{0}'.format(self.namez))
        self.change_time_slider(self.t_plot)
        self.change_z_slider(self.z_plot)
        self.set_grid()
```

```
        self.fig.canvas.draw()


    def set_grid(self):
        '''
        updates the plot calls from change_x_sliders
        '''
        z_cell = find(self.z[self.fc_idx] == self.z_plot).flatten()
        z_cell = z_cell.flatten()

#         print(z_cell.shape)
        data_t = self.data[z_cell, self.t_plot]
#         print('>>',data_t.shape)

        plot_x =  self.x[z_cell]
        plot_y =  self.y[z_cell]
        # data_at_time_t = self.data[z_cell, self.t_plot]
#         print(plot_x.shape)
        self.template.fill(0)
        self.template[self.x[self.z == self.z_plot] , self.y[self.z == self
        self.template[plot_x, plot_y] = data_t

        return self.template.T

    def plot(self):
        self.fig = \
                figure('zebra fish viewer {0}'.format(self.version))
        self.fig.canvas.set_window_title =\
                self.version


        self.ax = self.fig.add_subplot(111)
        # adjust axes to allow for new axes
        self.fig.subplots_adjust(left=0.25, bottom=0.3)

        # set nans to black
        cmap = matplotlib.cm.viridis
        cmap.set_bad('black', 1.)

        lim = .001
        vmin = -lim
        vmax = lim
        # draw image
        self.im = self.ax.imshow(self.template.T,\
                                 origin='upper',\
                                 cmap = cmap, \
                                 aspect = 'auto',\
                                 vmin = vmin,\
```

```
                                    vmax = vmax,\
                                    )
            [self.p] = self.ax.plot(self.x[0], self.y[0], marker = '$+$', c = '

            colorbar(self.im)

            # axis labels
            self.ax.set_xlabel('x')
            self.ax.set_ylabel('y')

#            self.ax.set_xlim([0, self.dims[0]])
#            self.ax.set_ylim([0, self.dims[1]])

            # construct sliders
            time_ax = self.fig.add_axes([0.25, 0.15, 0.65, 0.03])
            z_ax    = self.fig.add_axes([.25, .1, .65, .03])
            cell_ax = self.fig.add_axes([.25, .05, .65, .03])

            self.cell_slider = Slider(cell_ax, 'cell idx', 0, len(self.fc_idx),

            self.time_slider = Slider(\
            time_ax, 'time', 0, 40-1, valinit = self.t_plot)

            self.z_slider    = Slider(\
            z_ax, 'z-slice', 0, np.max(self.z[self.fc_idx]), valinit = self.z_p
            self.change_cell_slider(0)

            self.set_grid()




            plt.show()

            # set functions for value change of sliders
            self.time_slider.on_changed(self.change_time_slider)
            self.z_slider.on_changed(self.change_z_slider)
            self.cell_slider.on_changed(self.change_cell_slider)
            # register exit events
            def handle_close(evt):
                close(self.fig)
                self.alpha = 0
            # connect close_event to figure
            self.fig.canvas.mpl_connect('close_event', handle_close)
            # show figure
            # self.fig.show()
            # keep figurwith File('tmp.hdf5') as f:
```

```
        self.alpha = 1
#         while self.alpha:
#             # is this needed
#             pause(1e-10)

fv = fish_viewer()
# show(dummy.plot())
show(fv.plot())


#%%
#with File('tmp.hdf5') as f:
#    g = f['cross correlation'][str(9995)]
#    for i in g:
#        print(g[i])
#        assert 0
#
```

5 preliminary analysis script:

```
In [ ]: from h5py import File
        import numpy as np
        from pylab import *


        def create_or_get_group(file, name):
            try:
                group = file.create_group(name)
            except:
                group = file[name]
            return group


        def get_dataset(file, name, store_name = 0):
            with File(file) as f:
                group = f[name]
                out = []
                # print(group)
                for idx, member in enumerate(group):
                    out.append([])
                    # print(member)
                    if store_name:
                        save_this = [group[str(member)].value, member]
                    else:
                        save_this = group[str(member)].value
                    out[idx].append(save_this)
```

```
        return np.array(out).squeeze()

def comp_cross(idx, file):
    n_points = 10

    with File(file) as f:
        group = create_or_get_group(f, 'cross correlation')
        group_ft = create_or_get_group(f, 'filtered ft')
        for i in idx:
            group_cell = create_or_get_group(f, str(i))

            for j in idx:

                try:
                    group_cell.create_dataset(str(j), data = data_i)
                except:
                    pass

def comp_fft_filtered(file):
    try:
        File(file)['filtered fft']
        print('filtered fft present - exiting')
    except:
        with File(file) as f:
            group_filtered = f['filtered']
            group_fft_filtered = f.create_group('filtered fft')
            for idx, member in enumerate(group_filtered):
                print(idx)
                data_i = np.fft.fft(group_filtered[str(member)].value)
                group_fft_filtered.create_dataset(str(idx), data = data_i)

def put_mat_in_hdf5(file):
    '''
    Don't run this unless hdf5 file doesnt contain these values
    '''

    import scipy
    from  scipy.io import loadmat
    # del File(file)['coordinates']

    var = ['xx','yy','zz', 'frame_turn','cell_resp']
    data = scipy.io.loadmat('data.mat', variable_names = var)
    y = (data['xx'] - 1 )// 4
    x = (data['yy'] -1) // 4
    z = data['zz'] - 1

    print(np.max(x), np.max(y), np.max(z))
    cr = data['cell_resp']
```

```
    with File(file) as f:
        # del f['raw']
        del f['coordinates']
        group = f.create_group('coordinates')
        group.create_dataset('x', data = x)
        group.create_dataset('y', data = y)
        group.create_dataset('z', data = z)
        # group = f.create_group('raw')
        # for idx, i in enumerate(cr):
        #     group.create_dataset(str(i), data = i)

def create_template(file):
    with File(file) as f:
        coord = f['coordinates']
        x = coord['x'].value
        y = coord['y'].value
        z = coord['z'].value
        template = np.zeros((np.max(x)+1, np.max(y) + 1, np.max(z) + 1))
        template[x,y,z] = 1
        imshow(np.sum(template,2), aspect = 'auto')
        del f['template']
        group = f.create_group('template')
        f['template'].create_dataset('xy', data  = np.sum(template, 2))
        f['template'].create_dataset('yz', data  = np.sum(template, 0))
        f['template'].create_dataset('xz', data = np.sum(template,  1))
        f['template'].create_dataset('xyz',  data = template)

def get_frontal_cells(file):
    with File(file) as f:
        cord = f['coordinates']
        cord = np.array([cord[i].value for i in cord]).squeeze()

        # print(cord.shape); assert 0
        del f['front cells']
        f.create_group('front cells')
        print(cord.shape, np.max(cord,1))

        front_cell = np.logical_and(np.logical_and(cord[1,:] < 100, cord[0]
                                cord[2]< 20).flatten()
        idx_front_cell = find(front_cell == 1).flatten()
        f['front cells'].create_dataset('indices', data = idx_front_cell)
        f['front cells'].create_dataset('names', data = front_cell)


def filter_signal(file):

    # filter params
    freq = 1/6
```

```
        nF   = 1 / 2  # 1 hz sampling resolution assumed

        b, a = signal.butter(N = 10, Wn = freq/nF)
        with File(file) as f:
            try:
                group = f.create_group('filtered')
            except:
                group = f['filtered']
            group_norm = f['normalized']
            i = 0
            while True:
                try:
                    data = signal.filtfilt(b,a, group_norm[str(i)].value)
                    try:
                        group.create_dataset(str(i), data = data)
                    except:
                        pass
                except:
                    print('breaking loop')
                    break
                i += 1

    def cross_correlation(file):
        with File(file) as f:
            group = f['filtered fft']
            group_front = f['front cells']['indices'].value
            # del f['cc']
            group_cross =  f.create_group('cross correlation')
            fig, ax = subplots(1,1)
            ax.set_ylim([-1,1])
            n = 40
            [p] = ax.plot(range(n),range(n))

            for idx, i in enumerate(group_front):
                print('starting on {0} {1}'.format( idx, i))
                cell_data = group_cross.create_group(str(i))
                for jdx,  j in enumerate(group_front):
                    x = group[str(i)].value
                    y = group[str(j)].value
                    tmp = (real(np.fft.ifft( (x * np.conj(y)) / len(x) ) )) [:n

                    p.set_ydata(tmp)
                    ax.set_title('cell {0} > {1}'.format(i,j))

                    cell_data.create_dataset(str(j), data = tmp)
```

```
file1 = 'data.hdf5'
file2 = 'tmp.hdf5'
#
#with File(file1, 'r') as f:
#    with File(file2, 'w') as g:
#        for i in f:
#            try: f.copy(i, g)
#            except: pass
#

close('all')
#cross_correlation(file2)
fig, ax = subplots(1,1)
[p] = ax.plot(range(0, -40, -1), range(40))
#assert 0
#xlim([0, 40])
ylim([-1,1])
with File(file2) as f:
    gr = f['cross correlation']
    for cell in gr:
        for cellj in gr[cell]:
            p.set_ydata(gr[cell][cellj].value)
            pause(.01)
            title('cell {0} {1}'.format(cell, cellj))
#            assert 0
```

Due to the slow temporal nature, temporal filtering was used. This figure shows the same subset of cells as above. The following code was used to provide movies for the cross-correlation over time. This approach was abandoned due to the computational complexity of the data.

```
In [22]: from multiprocessing import Process, JoinableQueue
         from joblib import Parallel, delayed

         def saver(q):
             with File('test1.hdf5') as f:
                 group = f.create_group('test')
                 while True:
                     val  = q.get()
                     print(val);
                     if val is None: break
                     val, i = val
                     group.create_dataset(str(i), data = val)
                 q.task_done()
                 # Finish up
                 q.task_done()
```

```
        def tmp(x, y, i):

        #     f.create_group('0')

        #     x, y = arg
            x = np.fft.fft(x)
            y = np.fft.fft(y)
            out = np.real(np.fft.ifft(x * conj(y)) / nT)

        #     f.create_dataset(str(i), data = out)
            q.put((out,i))

        q = JoinableQueue()
        p = Process(target=saver, args=(q,))
        p.start()
        Parallel(n_jobs=2, verbose=0)(delayed(tmp)\
        (cr[i,:], cr[i,:],i) for i in range(10))
        q.put(None) # Poison pill
```

```
In [ ]: def run(cell, nT, good_cells, x_cord, y_cord, z_cord, file = file):
            '''visualizes the plot in 2D'''
            # open the hd5py file
            file = File(file)
            # get the 'connections' of the parsed cell
            data, names = get_sig_value(cell, nT, nC, file)

            # initiate movie writer
            FFMpegWriter = manimation.writers['ffmpeg']
            metadata = dict(\
                            title = 'Zebra fish data',\
                            artist='Casper van Elteren',
                            comment='Correlations bigger than abs(r) > .5')

            # obtain image dimensions
            dims = (np.max(x_cord), np.max(y_cord))
            writer = FFMpegWriter(fps = 60, metadata = metadata)

            # initialize figure

            fig = figure()
            ax = imshow([[0],[0]],\
                           interpolation = 'none',\
                           vmin = -.5, \
                           vmax = .5,\
                           cmap = 'seismic', \
                           extent = [0, dims[1], 0, dims[0]],\
                       aspect = 'auto')
            xlim([0, np.max(y_cord)])
```

```
    ylim([0, np.max(x_cord)])
    colorbar()
    sb.set_context('poster')

    # plot the source celll
    p = plot(y_cord[0, cell], x_cord[0, cell], 'k', markersize = 10,\
            marker = '$\plus$')
    save_to = 'correlation {0:d}.mp4'.format(cell)

    xlabel('x')
    ylabel('y')
    grid('off')
    #     %matplotlib qt5
#     assert 0
    # create movie
    with writer.saving(fig, save_to, 100):
        for t in range(nT):
#             clf()
#             if not t % 1000:
#                 print(t)
            #  obtain the data
            data_at_time_t = data[:,t]
            # cconvert to 3d
            data_at_time_t_3d = create_3d(\
                            cells_to_plot,\
                            x, \
                            y,\
                            z,\
                            data_at_time_t, names)[0]


            # convert to xy
            data_at_time_t_3d_xy = np.sum(data_at_time_t_3d, axis = 2)

            ax.set_data(data_at_time_t_3d_xy)
            title('time lag = {0:d}'.format(-t))
            # write to file
            writer.grab_frame()
    ans = []
    for i in range(10):
        ans.append(group[str(i)].value[:nT//2])

# run(biggest, nT, good_cells, x_cord,
    ans = []
    for i in range(10):
        ans.append(group[str(i)].value[:nT//2])y_cord, z_cord)
```