Neural Data Science

Lecturer: Prof. Dr. Philipp Berens

Tutors: Jonas Beck, Ziwei Huang, Rita González Márquez

Summer term 2022

Student name: Clarissa Auckenthaler & Marina Dittschar

Coding Lab 3

Not every cluster in the fitted mixture model corresponds to a single neuron's spikes (i.e. is a 'single unit'). We will explore different ways of identifying single units and telling them apart from multi unit activity. In all your plots, color-code the clusters consistently.

If needed, download the data file `nda_ex_2_*.npy` from ILIAS and save it in the subfolder `../data/` .

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
from scipy import correlate
from sklearn.decomposition import PCA
from scipy.io import loadmat
import copy
import itertools
from itertools import permutations

sns.set_style('whitegrid')
%matplotlib inline
```

Load data

In [2]:
```
m = np.load(r'data\nda_ex_2_meansnpy.sec')
S = np.load(r'data\nda_ex_2_covsnpy.sec')
a = np.load(r'data\nda_ex_2_labelsnpy.sec')
p = np.load(r'data\nda_ex_2_pisnpy.sec')

b = np.load(r'data\nda_ex_1_featuresnpy.sec')
t = np.load(r'data\nda_ex_1_spiketimes_tnpy.sec')
s = np.load(r'data\nda_ex_1_spiketimes_snpy.sec')
w = np.load(r'data\nda_ex_1_waveformsnpy.sec')
```

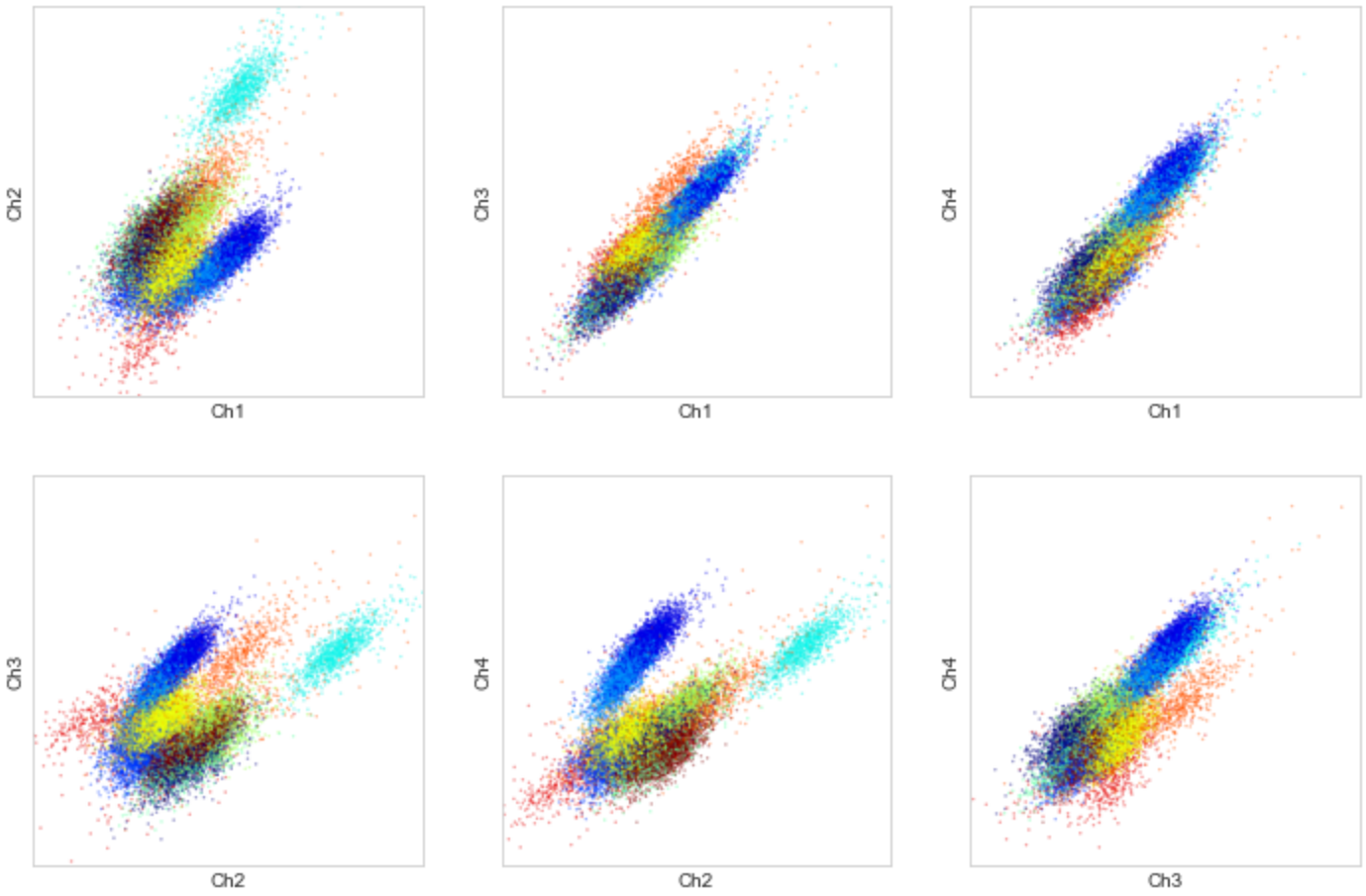
Task 1: Visual inspection of clusters

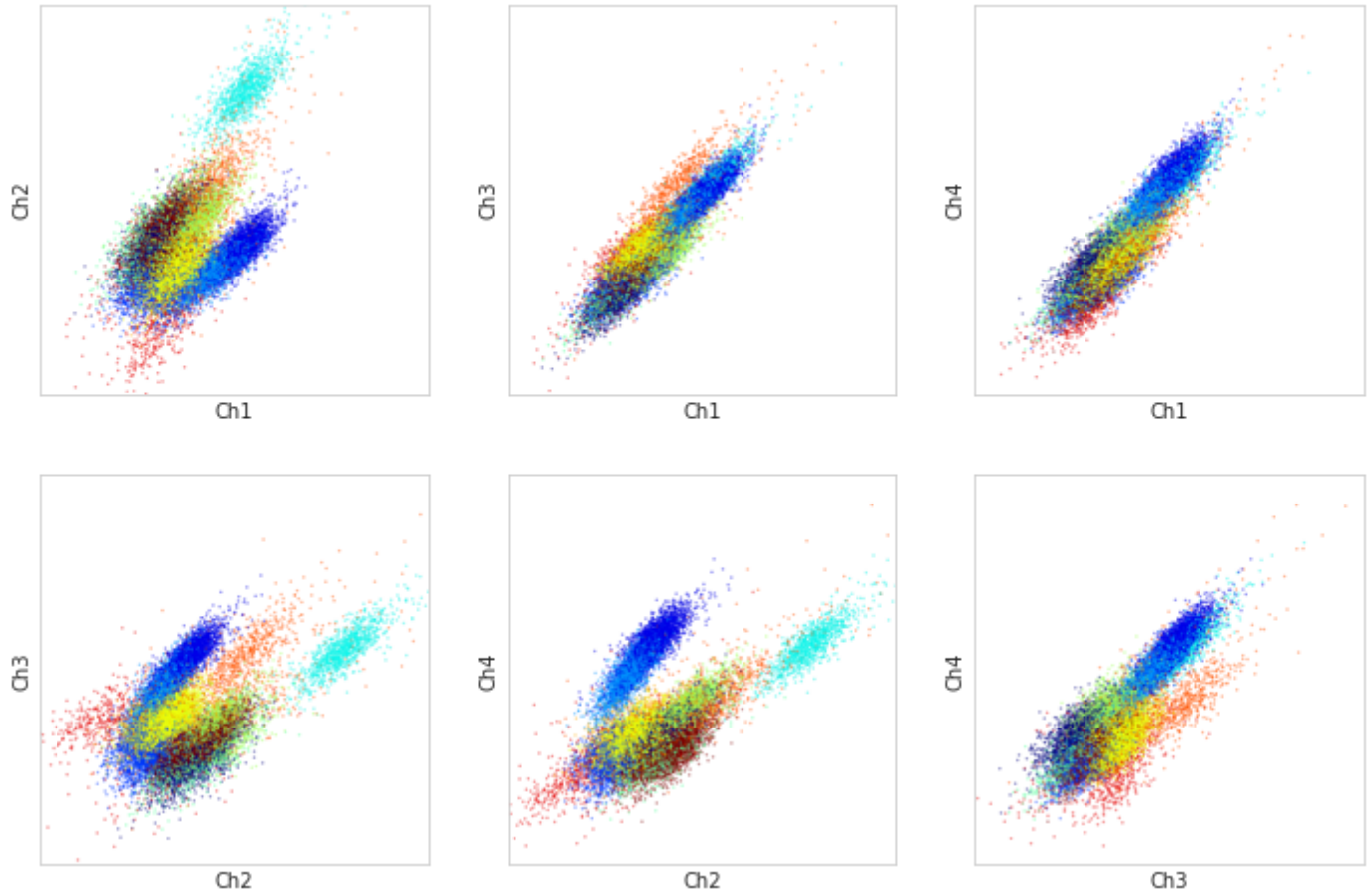
The most obvious candidates for single units are clusters that you can identify visually as being well separated from the rest. You can use the plotting function you implemented last week (Exercise 2 Task 4) as a first tool to identify putative single units. Of course there may be other less obvious cases in the data.

Grading: 0 pts

In [3]:

```
fig = plt.figure(figsize=(12, 8))

# insert your code here
colors = plt.cm.jet(np.linspace(0,1,max(a)+1))
c = np.squeeze(colors[a])
csum=[]

idx = [0, 3, 6, 9]
pl = 1

labels = ['Ch1','Ch2','Ch3','Ch4']
for i in np.arange(0,4):
    for j in np.arange(i+1,4):
        ax = plt.subplot(2,3,pl, aspect='equal')
        ax.scatter(b[:,idx[i]], b[:,idx[j]], c=c, s=.7,marker='.',alpha=0.5)
        csum= np.append(csum,c)
        plt.xlabel(labels[i])
        plt.ylabel(labels[j])
        plt.xlim((-800,1300))
        plt.ylim((-800,1300))
        ax.set_xticks([])
        ax.set_yticks([])
        pl = pl+1

plt.show()
```





Task 2: Visual inspection of waveforms

Plot the average waveforms and 100 examples from each cluster. This is a good sanity check and helps to identify potential artifacts such as electrical noise or clusters representing overlapping spikes (they often have large variance and few spikes).

Grading: 2 pts

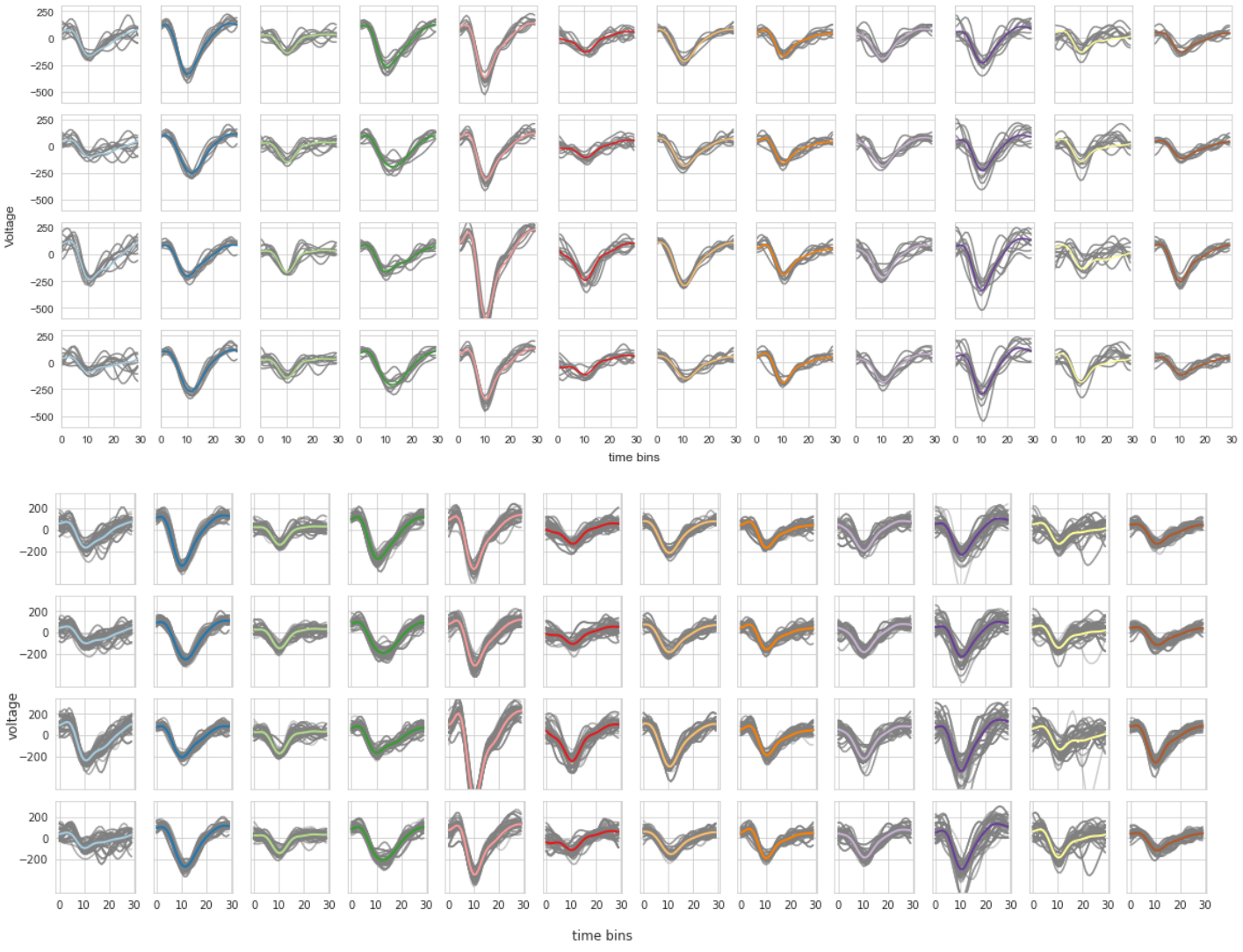
In [4]:
```
colors2 = plt.cm.Paired(np.linspace(0,1, 12))
c2 = np.squeeze(colors2[a])
```

In [5]:
```
def plot_waveforms(w, labels, N=100):

    '''Plot waveforms for all four channels of each cluster.
    Plot 100 sample waveforms, overlaid by the average.
    All panels are drawn on the same scale to facilitate comparison.

    Parameters
    ----------

    w: np.array, (len_waveform, n_samples, n_channels)
        Waveform.

    labels: np.array, (n_samples, )
        Cluster label for each datapoint.

    N: int, default = 100
        Number of waveforms to be plotted per cluster.

    '''
    # Insert your code here
```

```
    # ------------------------------------------------------------------
    # Plot 100 examples and the average waveform for each cluster (2 pts)
    # ------------------------------------------------------------------

    label_vals = np.unique(labels)

    fig, ax = plt.subplots(nrows= w.shape[2], ncols=label_vals.shape[0], sharex=True, shar
                           figsize=(16,6))

    fig.text(0.5, -0.01, 'time bins', ha='center', fontsize= 12)
    fig.text(-0.01, 0.5, 'Voltage', va='center', rotation='vertical',fontsize= 12)
    # plot waveforms for each channel and each cluster
    for i, label in enumerate(label_vals):
        a_cluster = w[:,a==label,:]
        for channel in np.arange(4):
                ax[channel, i].plot(a_cluster[:,:10,channel], c='gray', alpha=0.9)
                ax[channel, i].plot(a_cluster[:,:,channel].mean(axis=1), c=colors2[i], alp
                ax[channel, i].set_xlim(0,30)
                ax[channel, i].set_xticks([0,10,20,30])
                ax[channel, i].set_ylim(-600,300)

    plt.tight_layout()
    plt.show()
```

In [6]:

```
plot_waveforms(w, a)
```



Task 3: Auto/cross-correlation

Implement a function to calculate the auto/cross-correlograms of all clusters/pairs of clusters called `correlogram()`. Correlograms are useful mostly for two reasons:

- To identify clusters that represent multi unit activity. Neurons have a refractory period: after firing a spike they cannot fire another spike within a period of time (at least 1 ms, often more, depending on the cell type). Thus, if multiple cells contribute to one cluster, it won't have a refractory period.
- To identify two (or more) clusters that represent the same single unit: if this is the case the cross-correlogram of the two clusters should show the refractory period, since it consists of spikes from only one cell, which cannot occur too close to each other.

Plot a matrix with cross- and auto-correlograms like shown in the lecture. Use a bin size of 0.5 ms and a range of ±20 ms. Which auto-correlograms show a refractory period? Which cross-correlograms do?

You need to take some care to ensure that the implementation of the crosscorrelogram function is efficient, otherwise this may take a while.

Grading: 4 pts

In [7]:

```
def correlogram(t, labels, binsize, maxlag):

    '''Calculate cross correlogram.

    Parameters
    ----------

    t: np.array, shape=(n_spikes, )
        Spike times in samples.

    labels: np.array, shape=(n_spikes, )
        Cluster labels for each spike

    binsize: float
        Bin size.

    maxlag: int
        Maximal lag.


    Returns
    -------

    ccg: np.array, shape=(n_bins, n_clusters, n_clusters)
        computed correlograms

    all_bins: np.array, shape=(n_bins, )
        time bins relative to center
    '''

    # insert your code here

    # ------------------------------------------------
    # compute correlogram in a given timewindow (3 pts)
    # ------------------------------------------------
    # get cluster number
    cluster_no = np.unique(labels).shape[0]
    #compute bin-number
    bin_no = int(2*(maxlag/binsize) + 1)
    ccg = np.zeros((bin_no - 1, cluster_no,cluster_no))
```

```
    for ref_cluster in np.arange(cluster_no):
        for cluster in np.arange(cluster_no):
            bin = np.linspace(-maxlag, maxlag, bin_no)
            # define the windows for spikes
            wins = bin + t[labels==ref_cluster,np.newaxis]
            # get the spike times for cluster
            cluster_one = t[labels==cluster]

            counts = np.zeros(bin.shape[0]-1)
            all_counts_spike = np.zeros((0, bin.shape[0]-1))
            # for every spike, get the spikes in the reference window
            for i, val in enumerate(t[labels==ref_cluster]):
                counts_spike, bins_spike = np.histogram(cluster_one, wins[i])
                # add up the spikes
                counts = counts + counts_spike

            # append the spikes
            all_counts_spike = np.append(all_counts_spike, np.expand_dims(counts, axis=0),
            if ref_cluster == cluster:
                bin_middle = int(bin.shape[0]/2)
                # remove autocorellated spikes in the middle (results in large spike volur
                # visible in the suggested solution)
                all_counts_spike[:,bin_middle - 1: bin_middle+ 1] = 0

            ccg[:,ref_cluster, cluster] = all_counts_spike


    return ccg, bin
```

In [8]:

```
def plot_ccg(ccg, bins, figsize=(18,18)):

    '''Plot cross correlogram matrix.

    Parameters
    ----------

    ccg: np.array, shape=(n_bins, n_clusters, n_clusters)
        computed correlograms

    bins: np.array, shape=(n_bins, )
        time bins relative to center

    figsize: Tuple
        Set size of figure
    '''

    # insert your code here

    # ---------------------------
    # Plot crosscorelogram (1 pt)
    # ---------------------------
    fig, ax = plt.subplots(nrows= 12, ncols=12, sharex =True, figsize=figsize)
    val = 0
    fig.text(0.5, -0.01, 'delay (ms)', ha='center', fontsize= 12)
    fig.text(-0.01, 0.5, 'correlations', va='center', rotation='vertical',fontsize= 12)
    cluster_no = ccg.shape[1]
    # plot correlogram or each cluster
    for row in np.arange (cluster_no):
        for column in np.arange(cluster_no):
            if (column == row):
                #colorful if the cluster numbers are the same
                ax[row,column].hist(bins[:-1], bins, weights = ccg[:, row, column], color
            else:
```

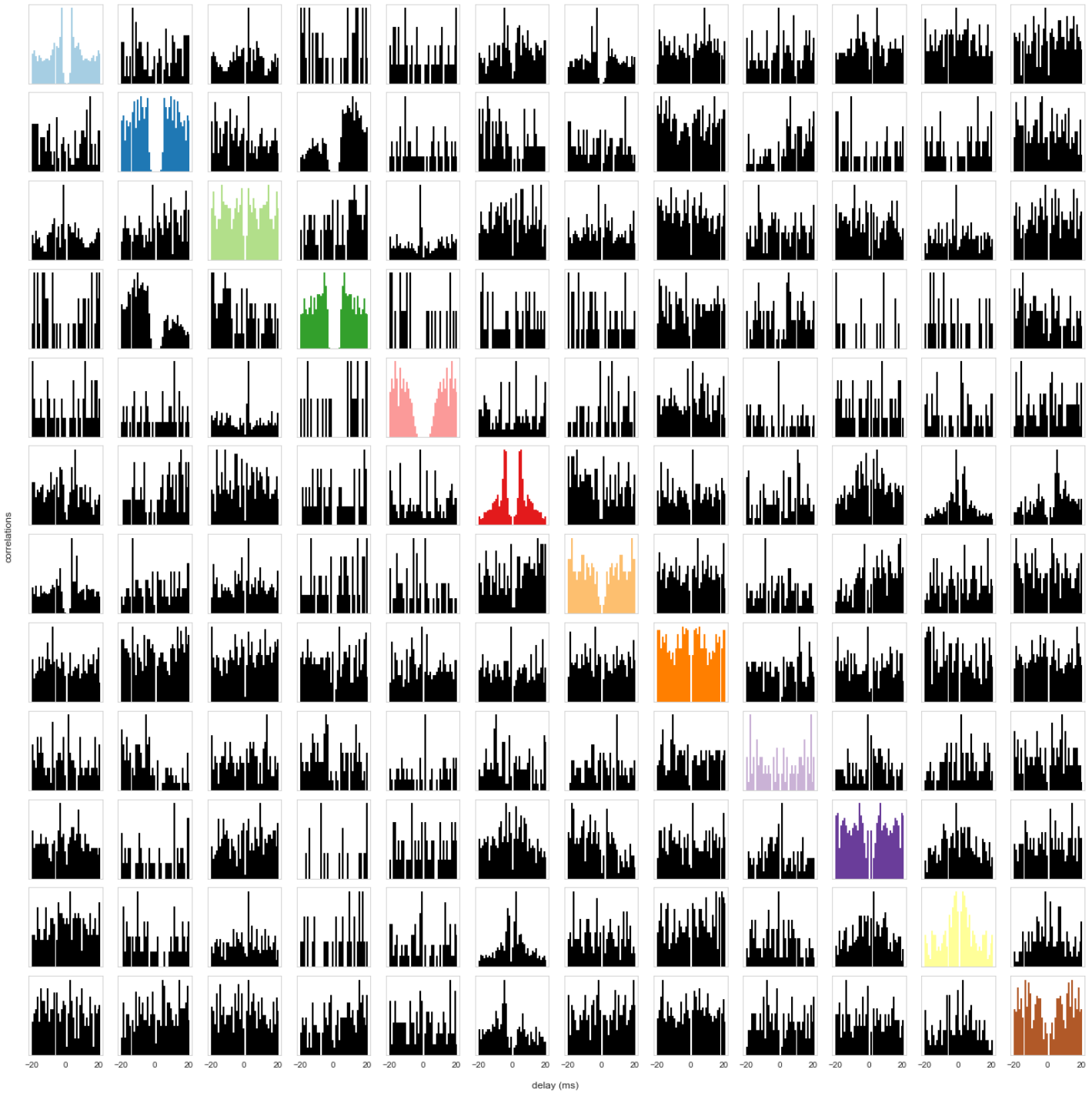
```
                ax[row,column].hist(bins[:-1], bins, weights = ccg[:, row, column], color

            ax[row][column].set_yticks([])
            ax[row][column].grid(False)

            val = val +1
    plt.tight_layout()
    plt.show()
```

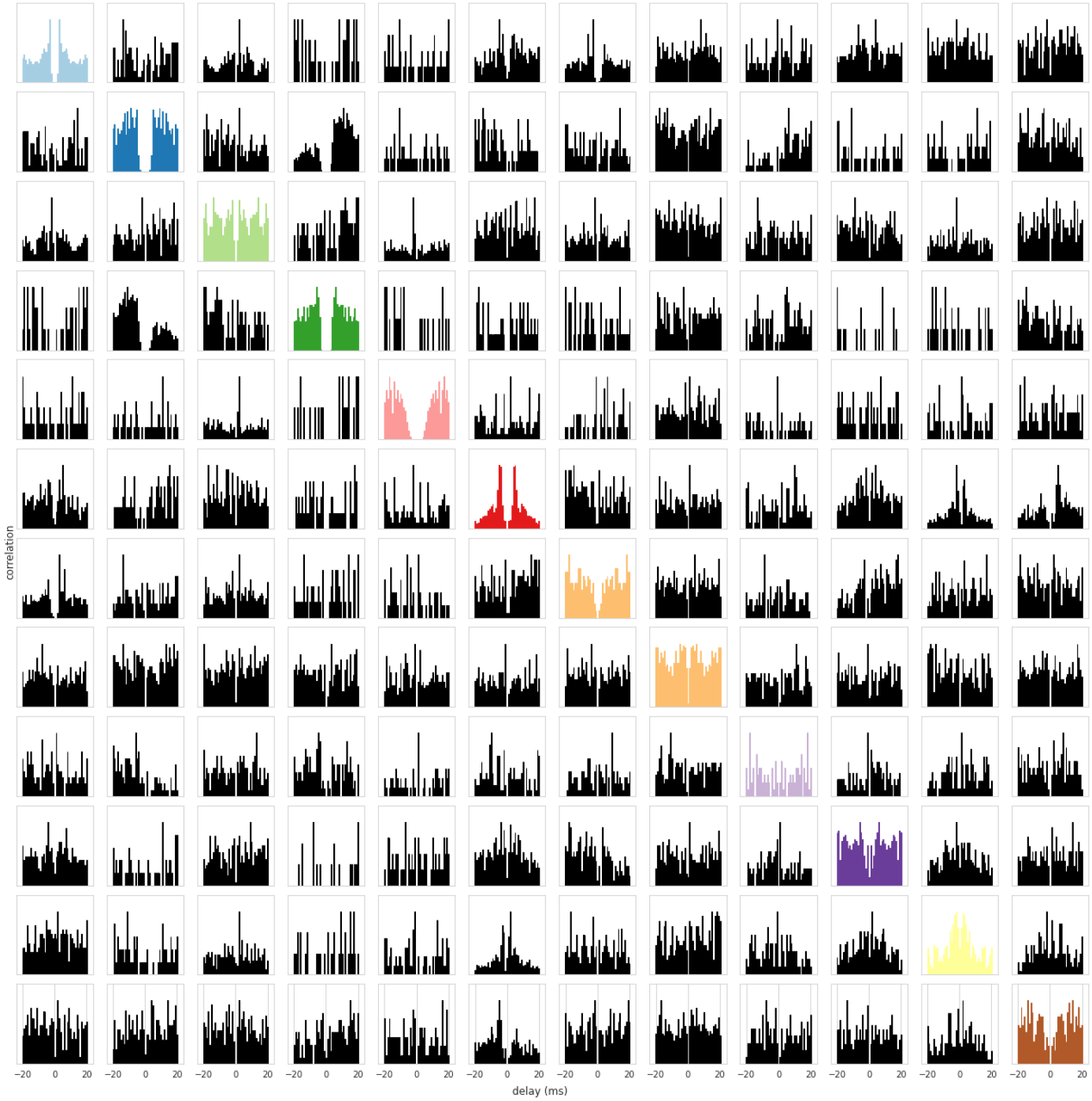
In [9]:

```
ccg, bins = correlogram(t, a, 0.5, 20)
```

In [10]:

```
plot_ccg(ccg, bins)
```





Task 4: Cluster separation

Implement linear discriminant analysis to visualize how well each cluster is separated from its neighbors in the high-dimensional space in the function `separation()`. Project the spikes of each pair of clusters onto the axis that optimally separates those two clusters.

Plot a matrix with pairwise separation plots, showing the histogram of the points in both clusters projected on the axis best separating the clusters (as shown in the lecture). *Hint:* Since Python 3.5+, matrix multiplications can be compactly written as `x@y`.

Grading: 4 pts

In [11]:

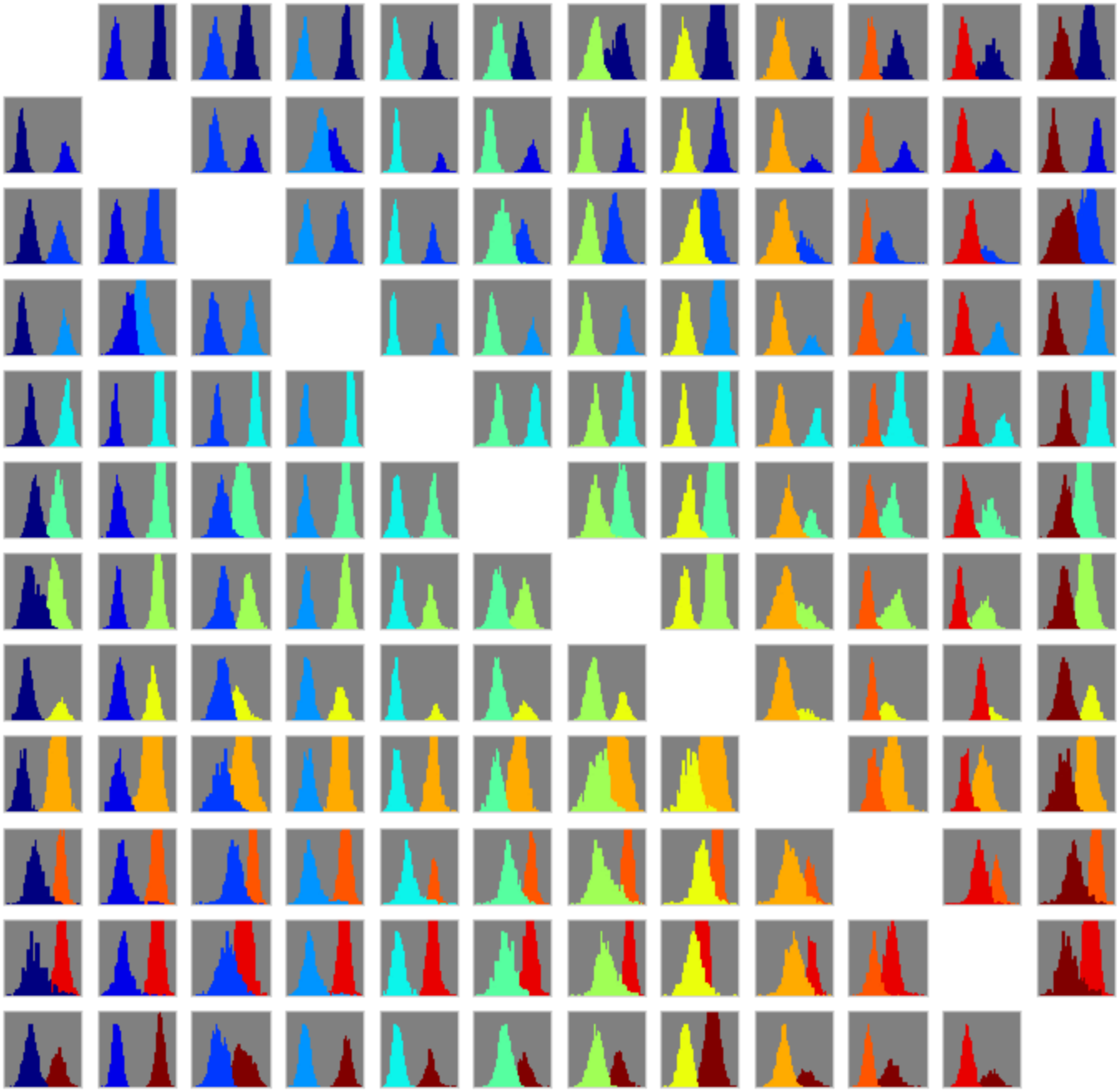
```
def separation(b, m, S, p, assignment, nbins=50, figure=3):
    '''Calculate cluster separation by LDA.

    proj, bins = separation(b, m, S, p, assignment)
```

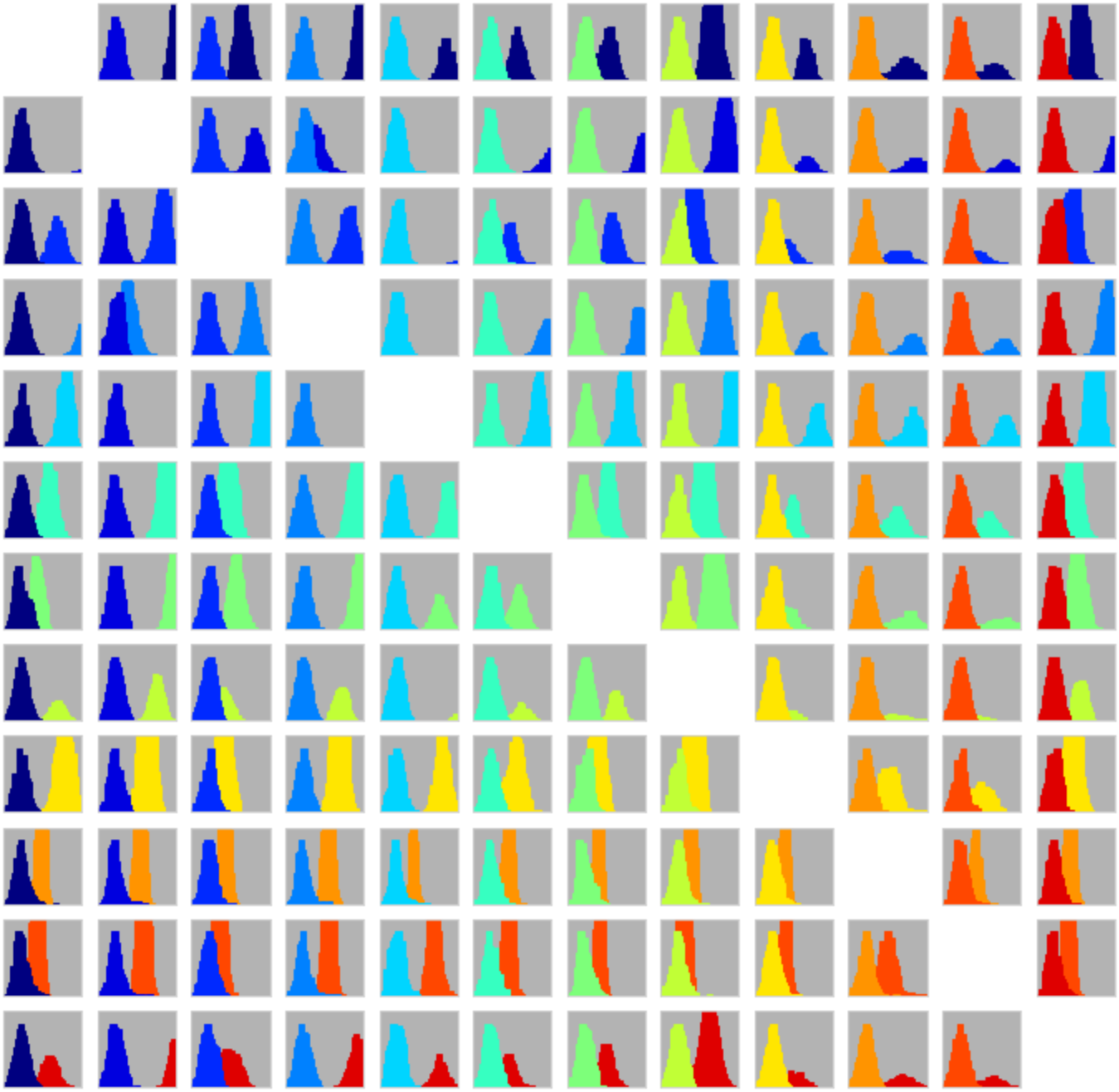
```
    projects the data on the LDA axis for all pairs of clusters. The result
    is normalized such that the left (i.e. first) cluster has
    zero mean and unit variances. The LDA axis is estimated from the model.
    ---

    Parameters
    ----------
    b: np.array, (n_spikes, n_features)
        Features.

    m: np.array, (n_clusters, n_features)
        Means.

    S: np.array, (n_features, n_features, n_clusters)
        Covariance.

    p: np.array, (n_clusters, )
        Cluster weight.

    nbins: int
    number of bins

    assignment: np.array, (n_spikes, )
        Cluster assignments / labels for each spike


    Returns
    -------

    proj: np.array, (n_bins, n_clusters, n_clusters)
        computed lda histo
# Comparing the cells in particular
# Each row of pixels in one of the subplots represents on row/column in the cluster sepera
# the colour codes for the height of the bar plots
compare_cell_pair(proj, [1,3], "Comparing Cell 1 and 3")
compare_cell_pair(proj, [5,6], "Comparing Cell 5 and 6")
compare_cell_pair(proj, [9,10], "Comparing Cell 9 and 10")grams   #bins x #clusters x #clu

    bins: np.array, (n_bins, n_clusters, n_clusters)
        bin times relative to center    #bins x 1
    '''

    # permutations for each cluster combination
    comp= list(map(list,permutations([0,1,2,3,4,5,6,7,8,9,10,11],2)))
    no_clusters = S.shape[1]
    fig, ax = plt.subplots(nrows= no_clusters, ncols=no_clusters, figsize=(12,12), squeeze
    bin_no = nbins
    counts_all = np.zeros((bin_no,no_clusters, no_clusters))
    bins_all = np.zeros((bin_no + 1,no_clusters, no_clusters))
    for i in np.arange(len(comp)):
    # -------------------------------------------------------------------------
    # compute the optimal separating axes for each pair of clusters (2 pts)
    # -------------------------------------------------------------------------
        cluster= comp[i]

        # get the covariances for the two clusters
        S_all = S[:,:,cluster[0]] + S[:,:,cluster[1]]
        # inverse of covariances
        S_inv = np.linalg.inv(S_all)
        # means for clusters
        m1 = m[cluster[0], :]
        m2 = m[cluster[1], :]
        # compute the difference
        mdiff = m2 - m1
        # projection vector
        W = np.dot(S_inv, mdiff)
```

```
        # get the cluster points
        points_cl1 = b[a==cluster[0]]
        points_cl2 = b[a==cluster[1]]
        # project
        y1 = np.dot(W, points_cl1.T)
        y2 = np.dot(W, points_cl2.T)

    # ----------------------------------------
    # normalise according to first cluster (1 pt)
    # ----------------------------------------
        # get the two histograms for the two clustes
        counts_ref, bins_ref = np.histogram(y1, bins=bin_no)
        counts_2, bins_2 = np.histogram(y1, bins=bin_no)
        counts_all[:,cluster[0], cluster[1]] = counts_ref
        counts_all[:,cluster[1], cluster[0]] = counts_2
        bins_all[:,cluster[0], cluster[1]] = bins_ref
        bins_all[:,cluster[1], cluster[0]] = bins_2
        # weight the plot by the maximum in the reference cluster
        max_y1 = np.max(counts_ref)


    # ---------------------------------------
    # plot histograms on optimal axis (1 pt)
    # ---------------------------------------
        # make the plots
        ax[cluster[0]][cluster[1]].hist(y2, bins=bin_no, histtype ='stepfilled',color= col
        ax[cluster[0]][cluster[1]].hist(y1, bins=bin_no,histtype ='stepfilled', color= col
        ax[cluster[0]][cluster[1]].set_xticks([])
        ax[cluster[0]][cluster[1]].set_yticks([])
        ax[cluster[0]][cluster[1]].set_ylim(0,1.2)
        ax[cluster[0]][cluster[1]].grid(False)
        ax[cluster[0]][cluster[1]].set_facecolor('gray')


    for j in np.arange(12):
        fig.delaxes(ax[j][j])
    plt.show()

    return counts_all, bins_all
```

In [12]:

```
proj, bins = separation(b,m,S,p,a)
```





Task 5: Identify putative single units.

Use all of the above tools to identify all putative single units in the dataset. Which ones are these and why?

The cross-correlogram indicate that cluster 1,3 and 4 describe single unit activity. However, the correlogram also reveals that cluster 1 and cluster 3 probably record the same cell since there is a refactory period visible in their cross-correlogram. Thus, we can identify the single units [1,3] and 4.

In [13]:

```
# insert your code here
```