matching_plot_constelation

May 1, 2017

In [1]:
```
load_ext autoreload
```

In [2]:
```
autoreload 2
```

In [24]:
```
import numpy as np
import pyphi
import random
import makegridslibrary as me
import matplotlib
import matplotlib.pyplot as plt
import itertools
from pypci import pci
import pickle
from tsne import tsne
import bitarray as bit
from IPython.core.display import display, HTML

def i_to_bitlist(o):
    if not o:
        return [0]
    shifts = list( range( int( np.ceil(np.log2(o)) + 1 ) ) )
    shifts.reverse() # little endian
    return [(o >> shift) & 1 for shift in shifts]
```

In [292]:
```
# Load results from matching_LB.py


# Old measure

# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrand
results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrandom

print('EMD results')

with open(results_filename, 'rb') as f:
    results = pickle.load(f)

big_results_bar = results[2]
```

```
big_results_shuffled = results[3]

# for network in big_results_bar:

network = 'Grid'

temps = list(big_results_bar[network].keys())

# temp = 0
temp = temps[1]

print(network)
print('\tT : ', temp)
print('\tfor the bar')
for input_stim in big_results_bar[network][temp]:
    print('\t', input_stim)
    this_result = big_results_bar[network][temp][input_stim]
    bar_emd_big_phi = this_result[0]
    bar_emd_concepts = this_result[1]

print('\tfor the shuffled')
for input_stim in big_results_shuffled[network][temp]:
    print('\t', input_stim)
    this_result = big_results_shuffled[network][temp][input_stim]
    shu_emd_big_phi = this_result[0]
    shu_emd_concepts = this_result[1]
print('done.')

# New measure

# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrand
# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrand
# results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrand
results_filename = '/data/nsdm/pyphi/fivenodes_barVSshuffled_gridVSrandom

print('\nNew cut + entropy distance results')

with open(results_filename, 'rb') as f:
    results = pickle.load(f)

big_results_bar_new = results[2]
big_results_shuffled_new = results[3]

# for network in big_results_bar_new:

temps = list(big_results_bar[network].keys())

# temp = 0
```

```
            temp = temps[2]

            print(network)
            print('\tT : ', temp)
            print('\tfor the bar')
            for input_stim in big_results_bar_new[network][temp]:
                print('\t', input_stim)
                this_result = big_results_bar_new[network][temp][input_stim]
                bar_new_big_phi = this_result[0]
                bar_new_concepts = this_result[1]

            print('\tfor the shuffled')
            for input_stim in big_results_shuffled_new[network][temp]:
                print('\t', input_stim)
                this_result = big_results_shuffled_new[network][temp][input_stim]
                shu_new_big_phi = this_result[0]
                shu_new_concepts = this_result[1]
            print('done.')
```

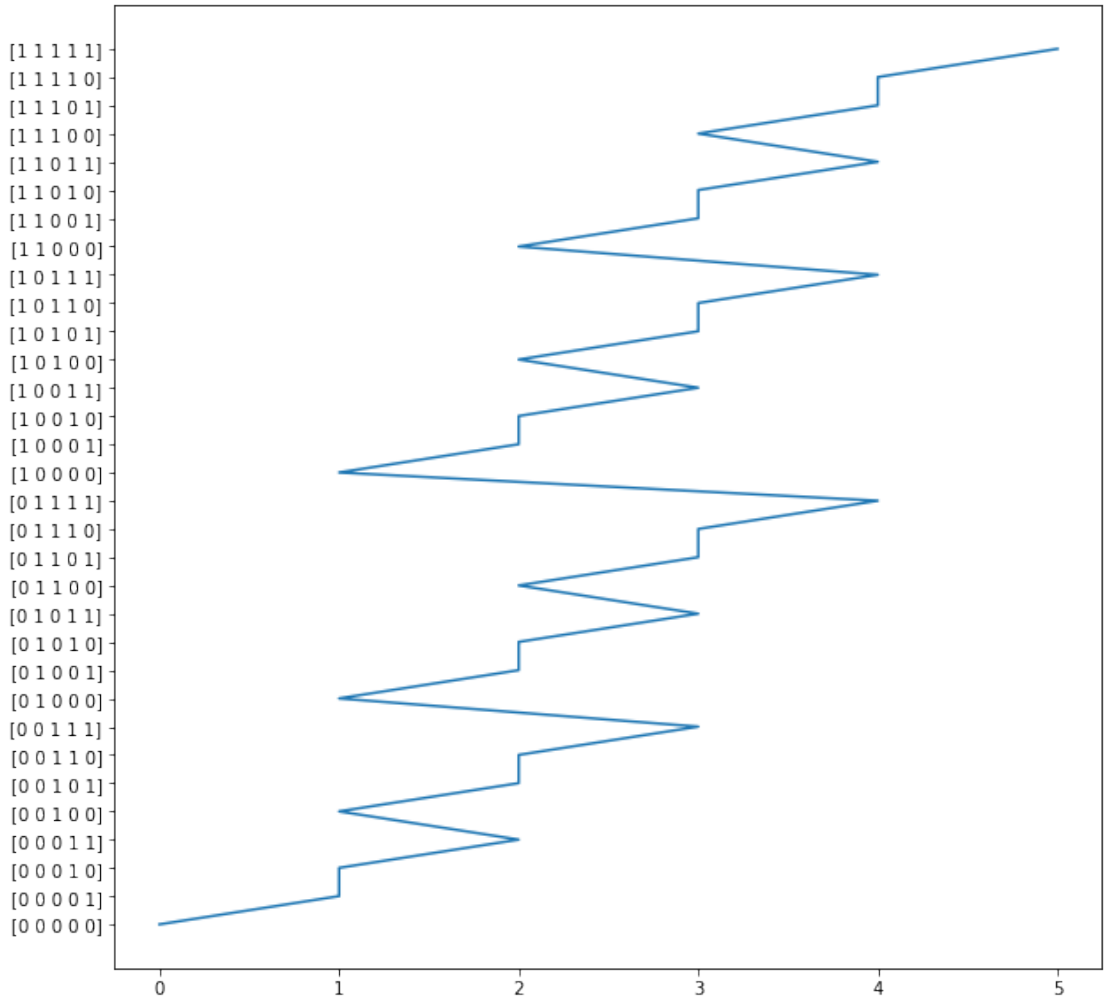
```
EMD results
Grid
        T :  0.1
        for the bar
         (0, 0, 0, 1, 1)
        for the shuffled
         (0, 0, 1, 0, 1)
done.

New cut + entropy distance results
Grid
        T :  0.8
        for the bar
         (0, 0, 0, 1, 1)
        for the shuffled
         (0, 0, 1, 0, 1)
done.
```

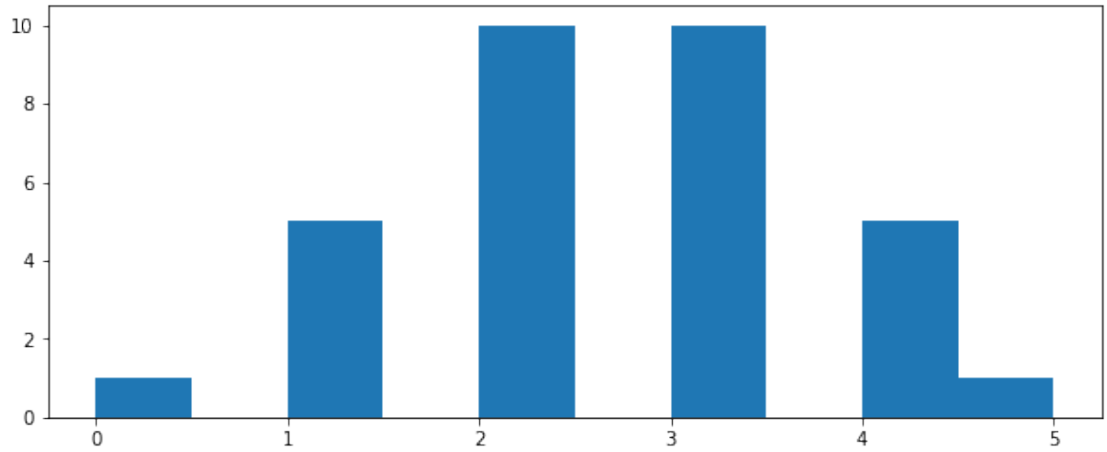
```
In [293]: # preprocess: build masks, labels, etc

            def copy_and_mask(X):
                Y = X.copy()
                Y_invalid = Y == -1
                Y[Y_invalid] = 0
                Y_big_invalid = np.any(Y_invalid, axis=0)
                return(Y, Y_invalid, Y_big_invalid)
```

```
# Use EMD
[bar_concepts, bar_big_phi, shu_concepts, shu_big_phi] = [bar_emd_concept

# Use new cuts and distances
# [bar_concepts, bar_big_phi, shu_concepts, shu_big_phi] = [bar_new_conce

(bar, bar_invalid, bar_big_phi_invalid) = copy_and_mask(bar_concepts)
(shu, shu_invalid, shu_big_phi_invalid) = copy_and_mask(shu_concepts)

N = int(np.ceil(np.log2(bar.shape[0])))

# Generate mechanisms labels
mechanisms_orders = np.array([sum(i_to_bitlist(m)) for m in range(2**N)])
# null mechanism is impossible
# mechanisms_orders = mechanisms_orders[1:]
# created a sorted index
mo_sorted_idx = np.argsort(mechanisms_orders)
sorted_mechanisms_orders = mechanisms_orders[mo_sorted_idx]

# Generate states labels
gs = np.array(list(itertools.product((0, 1), repeat=N)))
# print(grid_states)
# print(np.abs(np.diff(grid_states, axis=1)))
def neighbor(i,N,d):
    j = i + d
    # cycle right
    while j > N-1:
        j = j-N
    # cycle left
    while j < 0:
        j = j+N
    return j

def state_label_cont(s):
    if not np.sum(s) or np.sum(s) == len(s):
        label = 'full'
    else:
        second_order = [s[i] + s[neighbor(i,N,1)] for i in range(N)]
        n1 = np.sum(1*[s2 == 1 for s2 in second_order])
        n2 = np.sum(1*[s2 == 2 for s2 in second_order])
#         print(n2, s)
        if n2 == 0:
            label = 'shuffled'
        elif n2 == 1:
    #         if n1 > 0
            label = 'mixed'
        elif n2 == 2:
            label = 'bigbar2'
```

```
            elif n2 == 3:
                label = 'bigbar3'
            else:
                raise ValueError('Invalid state')
        return label

    def state_label_acti(s):
        s = [(s[i] + s[neighbor(i,N,1)])/2 for i in range(N)]
    #     s = [s[neighbor(i,N,-1) + s[i] + s[neighbor(i,N,1)] for i in range
    #     s = [1*(si > 1) for si in s]
    #     s = [1*(si != 1) for si in s]

    #     s = [1*((s[i] == 1) and (s[i] == s[neighbor(i,N,1)])) for i in rang
    #     s = [1*(s[i] == s[neighbor(i,N,1)]) for i in range(N)]
    #     s = [1*(s[neighbor(i,N,-1)] == s[i]) +  1*(s[neighbor(i,N,1)] == s

        return np.sum(s)

    # states_labels = [state_label_cont(s) for s in gs]
    states_labels = [state_label_acti(s) for s in gs]

    sl_sorted_idx = np.argsort(states_labels)
    sorted_states_labels = [states_labels[sl] for sl in sl_sorted_idx]



    plt.figure(figsize=(10,10))

    print(states_labels)
    plt.plot(states_labels, list(range(2**N)))
    plt.yticks(range(2**N), gs)
    plt.show()

    plt.figure(figsize=(10,4))
    plt.hist(states_labels)
    plt.show()

    # [np.sum(sorted_mechanisms_orders==o) for o in np.unique(mechanisms_orde
    # print(sl_sorted_idx)
    mechanisms_orders
    # print(mechanisms_orders)
    # print(sorted_mechanisms_orders)
    #print(sorted_states_labels)
```

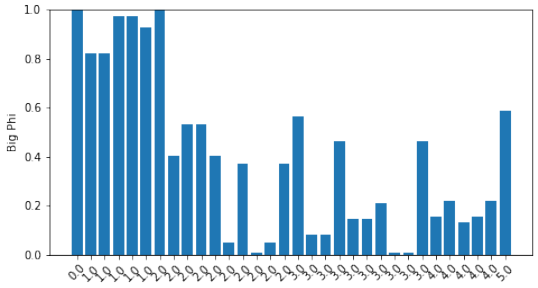
```
[0.0, 1.0, 1.0, 2.0, 1.0, 2.0, 2.0, 3.0, 1.0, 2.0, 2.0, 3.0, 2.0, 3.0, 3.0, 4.0, 1.
```

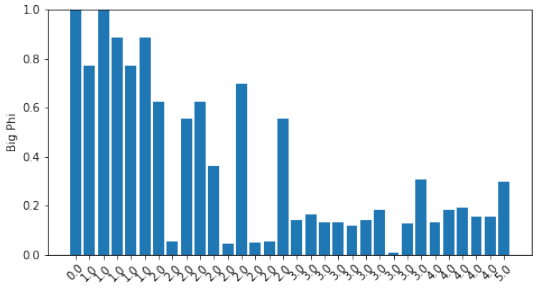

[1 1 1 1 1]
[1 1 1 1 0]
[1 1 1 0 1]
[1 1 1 0 0]
[1 1 0 1 1]
[1 1 0 1 0]
[1 1 0 0 1]
[1 1 0 0 0]
[1 0 1 1 1]
[1 0 1 1 0]
[1 0 1 0 1]
[1 0 1 0 0]
[1 0 0 1 1]
[1 0 0 1 0]
[1 0 0 0 1]
[1 0 0 0 0]
[0 1 1 1 1]
[0 1 1 1 0]
[0 1 1 0 1]
[0 1 1 0 0]
[0 1 0 1 1]
[0 1 0 1 0]
[0 1 0 0 1]
[0 1 0 0 0]
[0 0 1 1 1]
[0 0 1 1 0]
[0 0 1 0 1]
[0 0 1 0 0]
[0 0 0 1 1]
[0 0 0 1 0]
[0 0 0 0 1]
[0 0 0 0 0]
0
1
2
3
4
5

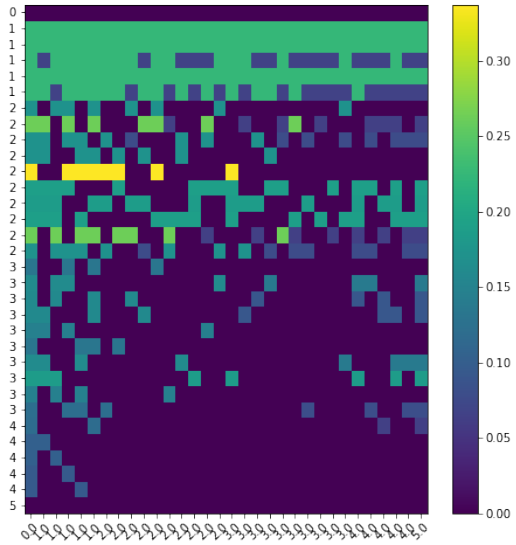

10
8
6
4
2
0
0
1
2
3
4
5

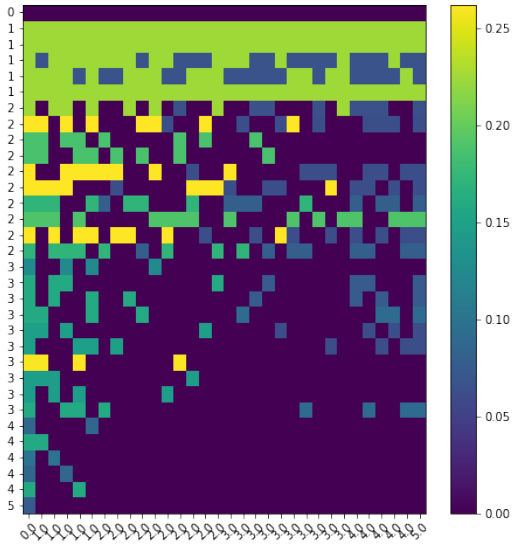
```
Out[293]: array([0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3,
                 4, 2, 3, 3, 4, 3, 4, 4, 5])

In [294]: # Big Phi

          plt.figure(figsize=(17,4))

          plt.subplot(1,2,1)
          plt.ylim([0, 1])
          Y = bar_big_phi.copy()
          Y[bar_big_phi_invalid] = 0
          plt.bar(range(len(Y)), Y[sl_sorted_idx])
          plt.xticks(range(len(Y)), sorted_states_labels, rotation=45)
          plt.ylabel('Big Phi')

          plt.subplot(1,2,2)
          plt.ylim([0, 1])
          # Y = bar_big_phi_new.copy()
          # Y[bar_big_phi_new_invalid] = 0
          Y = shu_big_phi.copy()
          Y[shu_big_phi_invalid] = 0
          plt.bar(range(len(Y)), Y[sl_sorted_idx])
          plt.xticks(range(len(Y)), sorted_states_labels, rotation=45)

          plt.ylabel('Big Phi')

          plt.show()



          # All concepts

          def norm_min_max(Y):
              Z = (Y-min(Y.flatten()))/(max(Y.flatten())-min(Y.flatten()))
              return Z

          plt.figure(figsize=(17,8))

          plt.subplot(1,2,1)
          # Y = X[sl_sorted_idx, :]

          # Why numpy, why...
          # Y = bar[mo_sorted_idx, sl_sorted_idx].copy()
          bar_sorted = bar[mo_sorted_idx, :].copy()[:, sl_sorted_idx]

          # bar_sorted[bar_invalid[mo_sorted_idx, :][:, sl_sorted_idx]] = 0
          # bar_sorted = np.ma.masked_where(X_invalid[sl_sorted_idx, :], X[sl_sorte
```

```
# bar_sorted = norm_min_max(Y)

plt.imshow(bar_sorted, aspect='auto')
plt.yticks(range(len(mo_sorted_idx)), sorted_mechanisms_orders)
plt.xticks(range(len(sl_sorted_idx)), sorted_states_labels, rotation=45)
# plt.xticks(range(len(sl_sorted_idx)), range(len(sl_sorted_idx)), rotati

plt.colorbar()

plt.subplot(1,2,2)
# Y = X_new[sl_sorted_idx, :]

# shu_sorted = shu[mo_sorted_idx, sl_sorted_idx].copy()
shu_sorted = shu[mo_sorted_idx, :].copy()[:, sl_sorted_idx]

# shu_sorted[shu_invalid[mo_sorted_idx, :][:, sl_sorted_idx]] = 0
# shu_sorted = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :], X_new

# shu_sorted = norm_min_max(Y_shu)

plt.imshow(shu_sorted, aspect='auto')
plt.yticks(range(len(mo_sorted_idx)), sorted_mechanisms_orders)
plt.xticks(range(len(sl_sorted_idx)), sorted_states_labels, rotation=45)
# plt.xticks(range(len(sl_sorted_idx)), range(len(sl_sorted_idx)), rotati
plt.colorbar()



# plt.figure(figsize=(17,8))

# Z = Y-Y_new
# # Z = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :] | ((Y_new[sl_s
# # Z = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :], Z)
# Z = np.ma.masked_where(X_new_invalid[sl_sorted_idx, :] | (Y[sl_sorted_i

# plt.imshow(Z, aspect='auto',cmap='bwr')
# plt.yticks(range(len(labels)), labels[sl_sorted_idx])
# plt.clim([-1, 1])
# plt.colorbar()

plt.show()
```









In [295]:
```
# number_only = 1
# number_only = 0

for number_only in range(2):

    display(HTML("<center><h1 style='font-size: %dpx;height: 100;'>%s</h1
            ((40, '# of concepts') if number_only else (50, '&Sigma; &phi

    # unique mechanisms
    unique_mechanism_orders = np.unique(sorted(sorted_mechanisms_orders))

    # unique and states
    unique_states_labels = np.unique(sorted(sorted_states_labels))
    nstates = len(unique_states_labels)

    # BAR
```

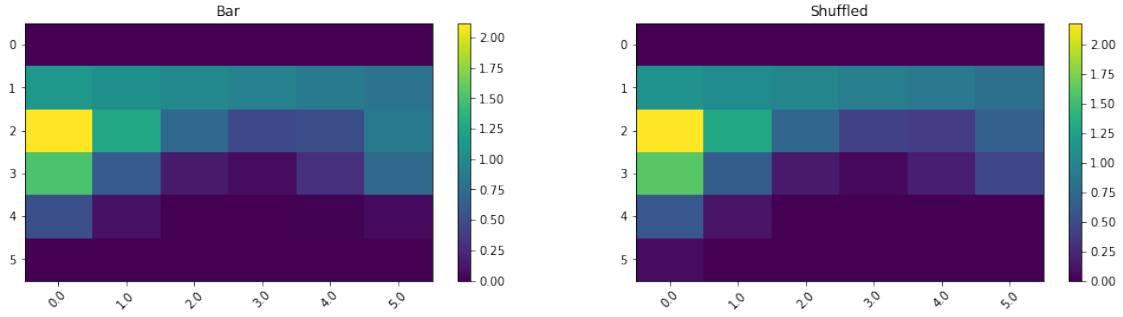
```
# find all the concepts for the bar
if number_only:
    bar_sorted_n = 1. * (bar_sorted > 0)
else:
    bar_sorted_n = bar_sorted

# sum all the mechanisms for a given order, per state
bar_sorted_cpmo = np.array([np.sum(bar_sorted_n[sorted_mechanisms_orc
                            for o in unique_mechanism_orders])

# remove invalid states
invalid_bar_states = np.all(bar_invalid[mo_sorted_idx, :][:, sl_sorte
sorted_bar_states_labels_noinv = np.delete(sorted_states_labels, np.w
bar_sorted_cpmo_noinv = np.delete(bar_sorted_cpmo, np.where(invalid_b

# take the mean and std accross valid states
bar_sorted_cpmosc = np.array([np.mean(bar_sorted_cpmo_noinv[:, [ss ==
                        for s in unique_states_labels]).T
bar_sorted_cpmosc_std = np.array([np.std(bar_sorted_cpmo_noinv[:, [ss
                        for s in unique_states_labels]).T

# SHUFFLED

# find all the concepts for the shuffled
if number_only:
    shu_sorted_n = 1. * (shu_sorted > 0)
else:
    shu_sorted_n = shu_sorted

# sum all the mechanisms for a given order, per state
shu_sorted_cpmo = np.array([np.sum(shu_sorted_n[sorted_mechanisms_orc
                            for o in np.unique(sorted(mechanisms_orde

# remove invalid states
invalid_shu_states = np.all(shu_invalid[mo_sorted_idx, :][:, sl_sorte
sorted_shu_states_labels_noinv = np.delete(sorted_states_labels, np.w
shu_sorted_cpmo_noinv = np.delete(shu_sorted_cpmo, np.where(invalid_s

# take the mean and std accross valid states
shu_sorted_cpmosc = np.array([np.mean(shu_sorted_cpmo_noinv[:, [ss ==
                        for s in unique_states_labels]).T
shu_sorted_cpmosc_std = np.array([np.std(shu_sorted_cpmo_noinv[:, [ss
                        for s in unique_states_labels]).T

plt.figure(figsize=(17,4))

plt.subplot(1,2,1)
plt.imshow(bar_sorted_cpmosc, aspect='auto')
```

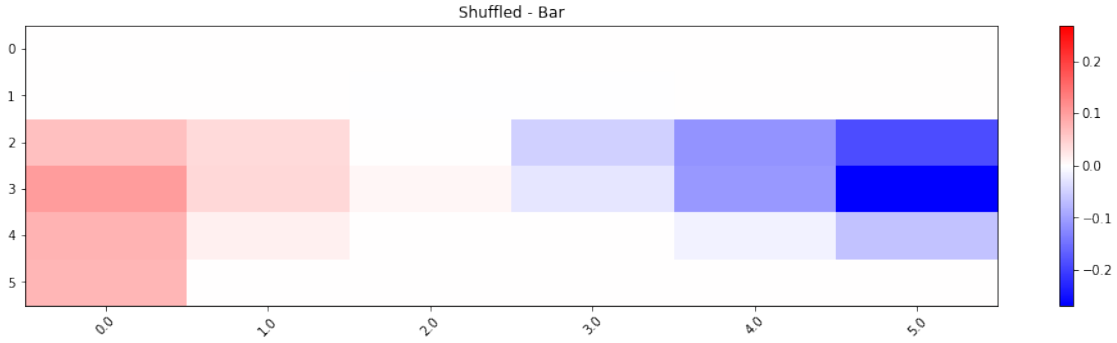
```
plt.xticks(range(len(unique_states_labels)), unique_states_labels, ro
plt.colorbar()
plt.title('Bar')

plt.subplot(1,2,2)
plt.imshow(shu_sorted_cpmosc, aspect='auto')
plt.xticks(range(len(unique_states_labels)), unique_states_labels, ro
plt.colorbar()
plt.title('Shuffled')

plt.show()
```

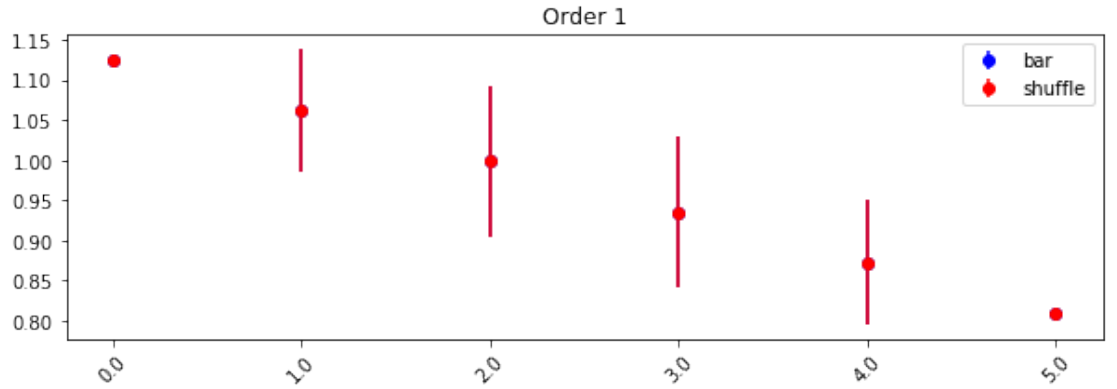
```
plt.figure(figsize=(17,4))
the_diff = shu_sorted_cpmosc-bar_sorted_cpmosc
the_diff_max = max(abs(the_diff.flatten()))
plt.imshow(the_diff, aspect='auto', cmap='bwr', vmin=-the_diff_max, v
plt.colorbar()
plt.xticks(range(len(unique_states_labels)), unique_states_labels, ro
plt.title('Shuffled - Bar')
plt.show()
```

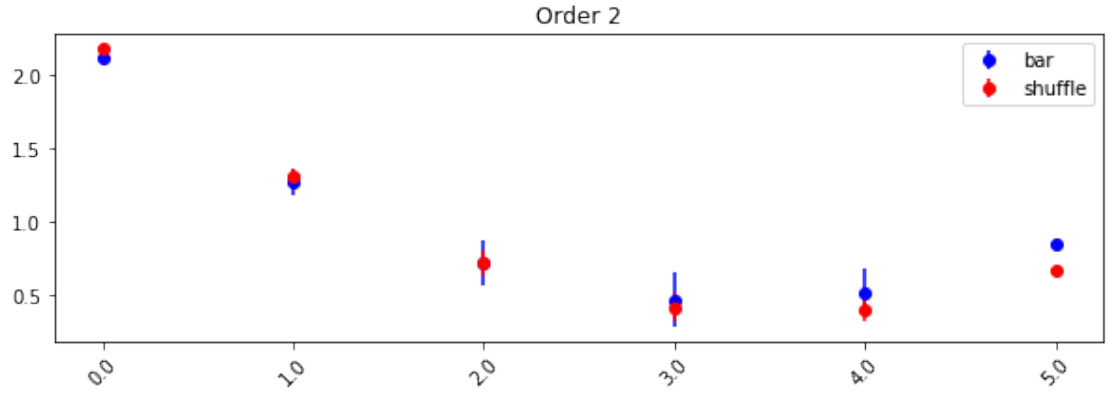
```
for o in range(1, len(unique_mechanism_orders)):

    plt.figure(figsize=(10,3))

    hb = plt.errorbar(range(nstates), bar_sorted_cpmosc[o, :], yerr=b
    hs = plt.errorbar(range(nstates), shu_sorted_cpmosc[o, :], yerr=s
    plt.legend([hb, hs], ['bar', 'shuffle'])
    plt.xticks(range(len(unique_states_labels)), unique_states_labels
    plt.title('Order %d' % o)

    plt.show()
```

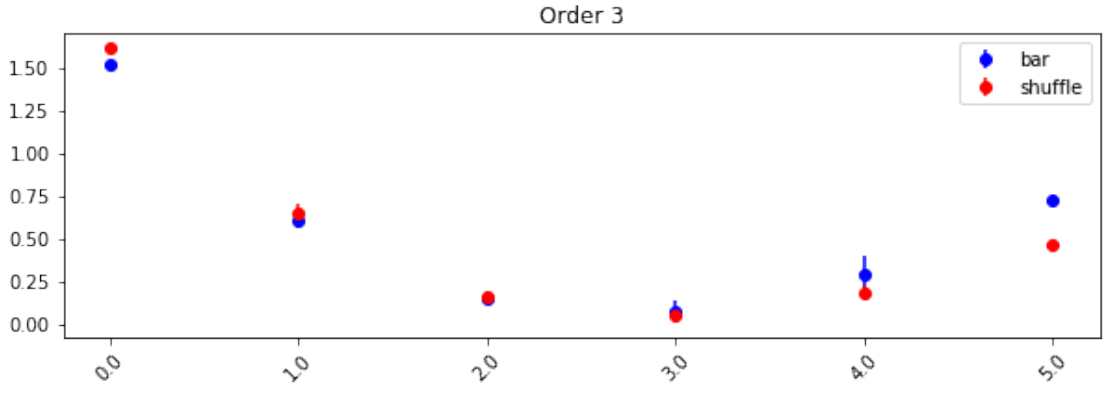
```
<IPython.core.display.HTML object>
```

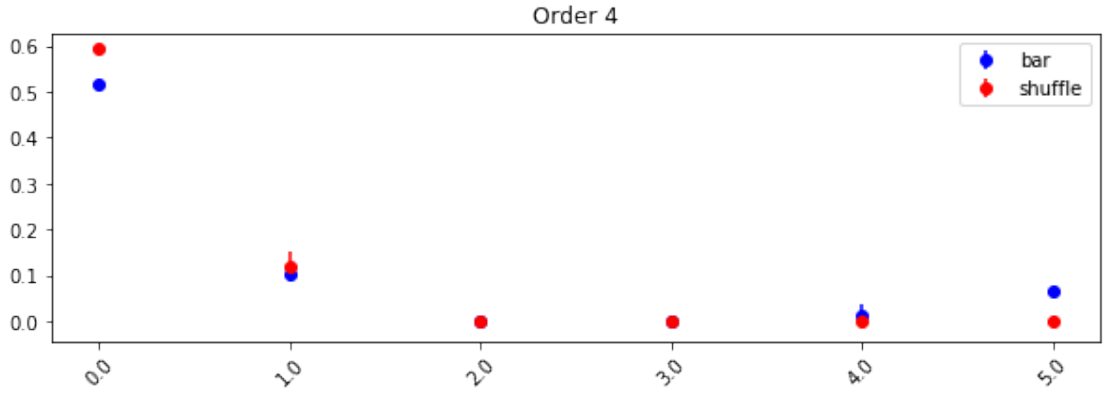


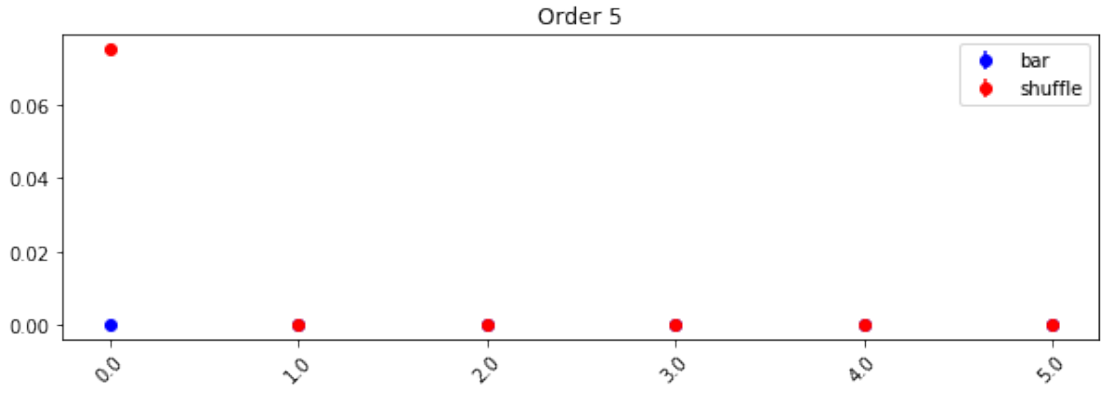

Shuffled - Bar
0
1
2
3
4
5
0.0
1.0
2.0
3.0
4.0
5.0
0.2
0.1
0.0
−0.1
−0.2


Order 1
1.15
1.10
1.05
1.00
0.95
0.90
0.85
0.80
0.0
1.0
2.0
3.0
4.0
5.0
bar
shuffle

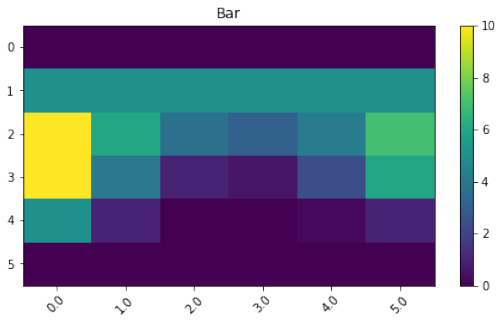

Order 2
2.0
1.5
1.0
0.5
0.0
1.0
2.0
3.0
4.0
5.0
bar
shuffle

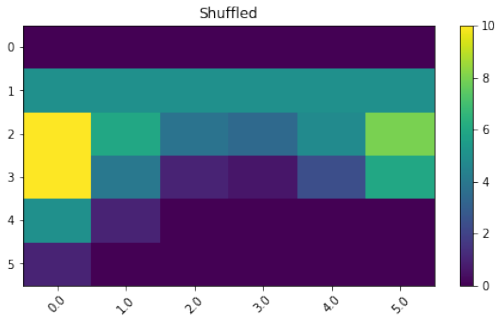


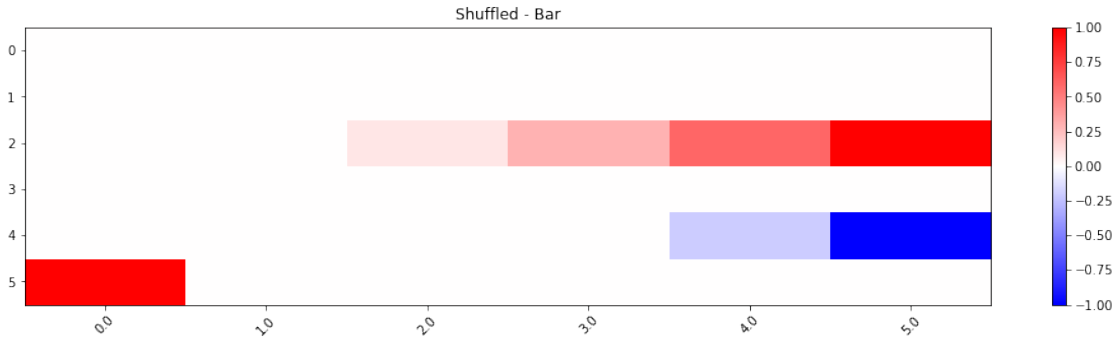


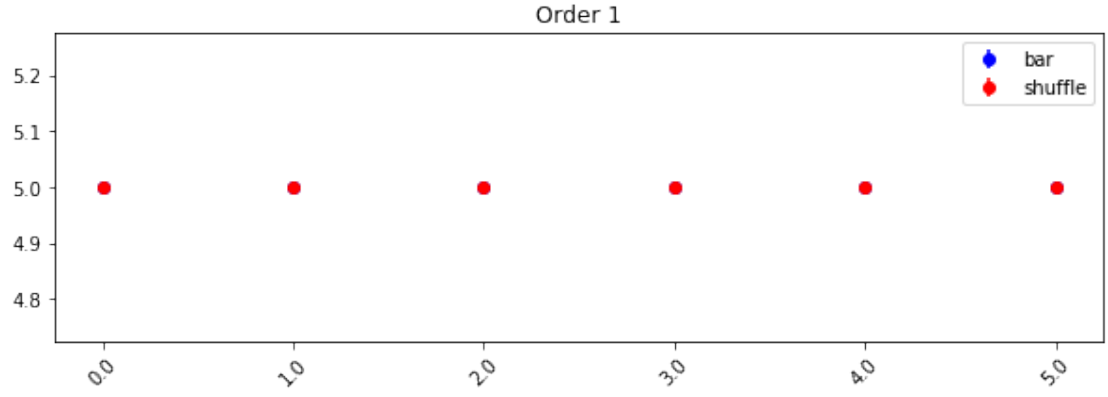


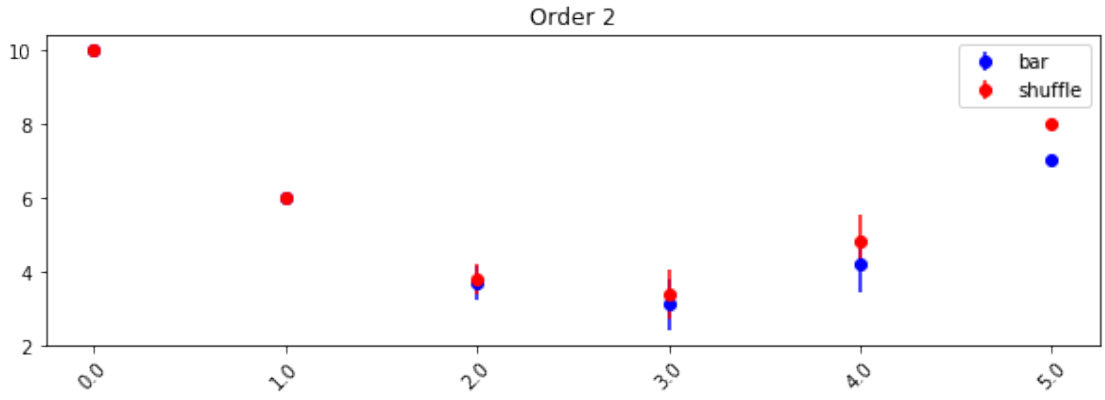

```
<IPython.core.display.HTML object>
```

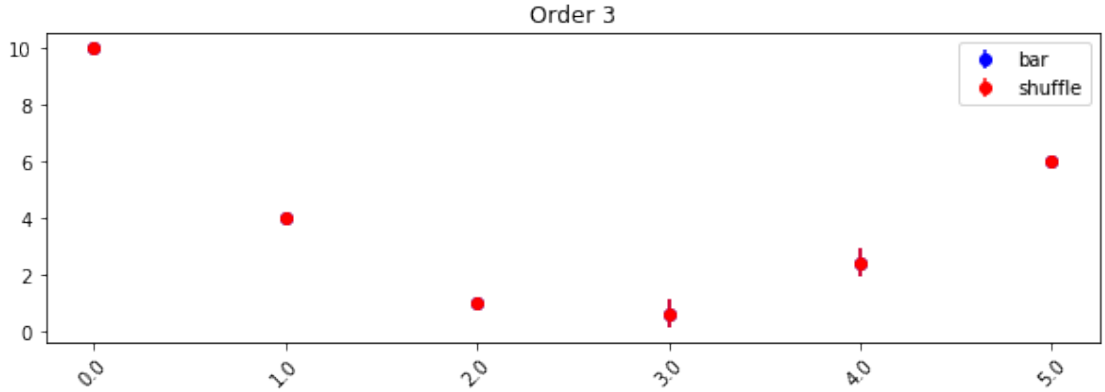

Bar
0
1
2
3
4
5
0.0
1.0
2.0
3.0
4.0
5.0
10
8
6
4
2
0

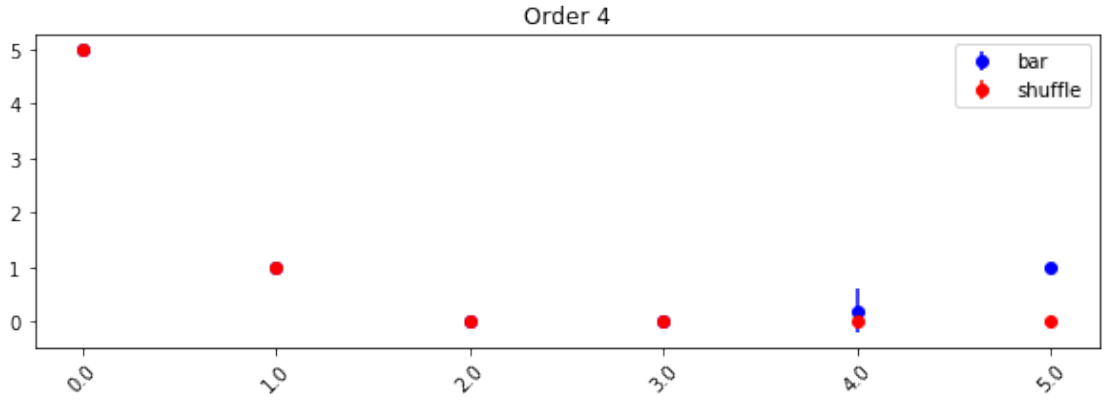

Shuffled
0
1
2
3
4
5
0.0
1.0
2.0
3.0
4.0
5.0
10
8
6
4
2
0

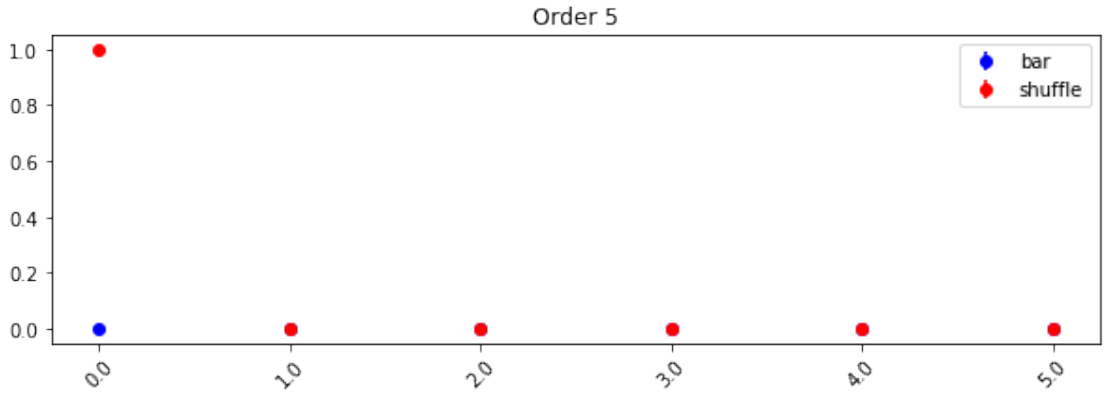

Shuffled - Bar
0
1
2
3
4
5
0.0
1.0
2.0
3.0
4.0
5.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Order 1
bar
shuffle
5.2
5.1
5.0
4.9
4.8
0.0
1.0
2.0
3.0
4.0
5.0


Order 2
bar
shuffle
10
8
6
4
2
0.0
1.0
2.0
3.0
4.0
5.0


Order 3
bar
shuffle
10
8
6
4
2
0
0.0
1.0
2.0
3.0
4.0
5.0


Order 4
bar
shuffle
5
4
3
2
1
0
0.0
1.0
2.0
3.0
4.0
5.0



In [296]:
```
# State labels

X = bar.T.copy()

# continuity
labels = states_labels.copy()
unique_labels_values = np.unique(labels)
unique_labels = unique_labels_values

# bar vs shuffled
# labels = states_labels.copy()
# labels_to_int = {'full': 1, 'shuffled': 2, 'mixed': 3, 'bigbar2': 4, 'b
# labels = np.array([labels_to_int[s] for s in labels])
# unique_labels_values = list(labels_to_int.values())
# unique_labels = labels_to_int.keys()

# Mechanism labels

# X = bar.copy()
# labels = mechanisms_orders.copy()
# unique_labels_values = np.unique(labels)
# unique_labels = unique_labels_values

# X = np.loadtxt("./tsne/mnist2500_X.txt");
# labels = np.loadtxt("./tsne/mnist2500_labels.txt");

Y = tsne.tsne(X, 2, 50, 20.0)
```

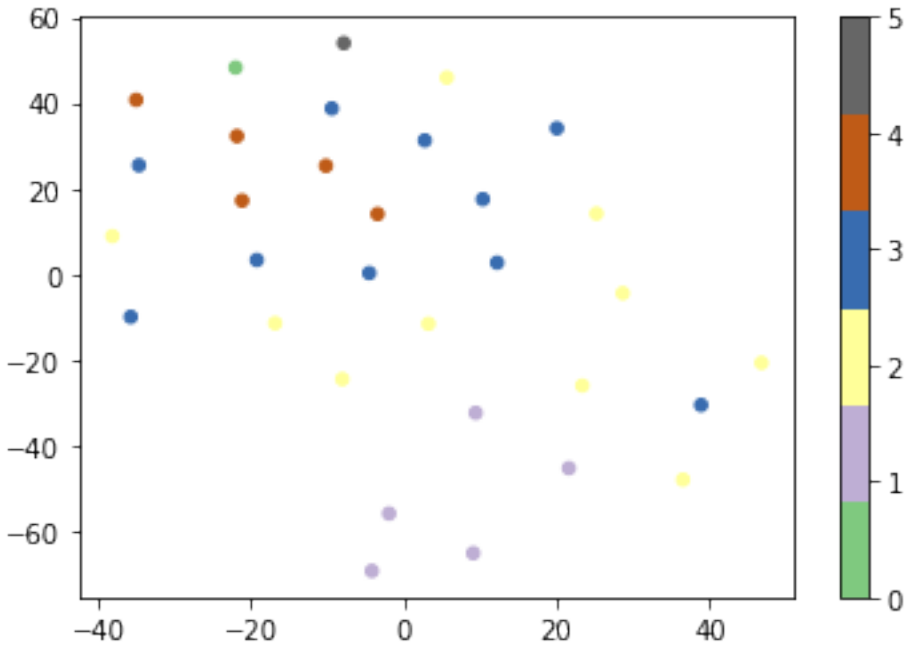
```
Preprocessing the data using PCA...
Computing pairwise distances...
Computing P-values for point  0  of  32 ...
Mean value of sigma:  0.291203847398
Iteration  10 : error is  10.4837432959
```

```
Iteration  20 : error is  11.4819187189
Iteration  30 : error is  11.5088365801
Iteration  40 : error is  10.8675816679
Iteration  50 : error is  12.6566282452
Iteration  60 : error is  12.2669640621
Iteration  70 : error is  12.5370034172
Iteration  80 : error is  15.2399334747
Iteration  90 : error is  12.3343903629
Iteration  100 : error is  11.9775214802
Iteration  110 : error is  1.47510849519
Iteration  120 : error is  1.28319780412
Iteration  130 : error is  1.13956329188
Iteration  140 : error is  1.0635813955
Iteration  150 : error is  1.017112156
Iteration  160 : error is  0.967191221603
Iteration  170 : error is  0.922606496994
Iteration  180 : error is  0.889502308391
Iteration  190 : error is  0.863720536523
Iteration  200 : error is  0.837044865693
Iteration  210 : error is  0.807745921611
Iteration  220 : error is  0.785551546804
Iteration  230 : error is  0.773435814399
Iteration  240 : error is  0.768229300014
Iteration  250 : error is  0.76407186042
Iteration  260 : error is  0.759023539633
Iteration  270 : error is  0.752162541608
Iteration  280 : error is  0.742947846965
Iteration  290 : error is  0.732107822125
Iteration  300 : error is  0.722311383481
Iteration  310 : error is  0.714545361652
Iteration  320 : error is  0.707838143063
Iteration  330 : error is  0.69879104601
Iteration  340 : error is  0.688883806642
Iteration  350 : error is  0.682600388086
Iteration  360 : error is  0.676294819
Iteration  370 : error is  0.669238766655
Iteration  380 : error is  0.661320193831
Iteration  390 : error is  0.651361538622
Iteration  400 : error is  0.643003519204
Iteration  410 : error is  0.637381542116
Iteration  420 : error is  0.630199378781
Iteration  430 : error is  0.620527223912
Iteration  440 : error is  0.610511951177
Iteration  450 : error is  0.599448285453
Iteration  460 : error is  0.584539612642
Iteration  470 : error is  0.565338930593
Iteration  480 : error is  0.539659669895
Iteration  490 : error is  0.523721545844
```

```
Iteration  500 : error is  0.518445998296
Iteration  510 : error is  0.515129803881
Iteration  520 : error is  0.511866187868
Iteration  530 : error is  0.508482878171
Iteration  540 : error is  0.505886455838
Iteration  550 : error is  0.50351269806
Iteration  560 : error is  0.500560811535
Iteration  570 : error is  0.494784952461
Iteration  580 : error is  0.490009738407
Iteration  590 : error is  0.487352791126
Iteration  600 : error is  0.485490607686
Iteration  610 : error is  0.483885019052
Iteration  620 : error is  0.482294808661
Iteration  630 : error is  0.480547158049
Iteration  640 : error is  0.478555852705
Iteration  650 : error is  0.476197741198
Iteration  660 : error is  0.473305845478
Iteration  670 : error is  0.46999605184
Iteration  680 : error is  0.466608677671
Iteration  690 : error is  0.463230633905
Iteration  700 : error is  0.45974163059
Iteration  710 : error is  0.455762186075
Iteration  720 : error is  0.450214406596
Iteration  730 : error is  0.4436018551
Iteration  740 : error is  0.440112139956
Iteration  750 : error is  0.437185478045
Iteration  760 : error is  0.433439918092
Iteration  770 : error is  0.428451848277
Iteration  780 : error is  0.422126169432
Iteration  790 : error is  0.417646053336
Iteration  800 : error is  0.414966778128
Iteration  810 : error is  0.413094606642
Iteration  820 : error is  0.411406270597
Iteration  830 : error is  0.40968676823
Iteration  840 : error is  0.407859310726
Iteration  850 : error is  0.405927126821
Iteration  860 : error is  0.403940743924
Iteration  870 : error is  0.401888448374
Iteration  880 : error is  0.399594259312
Iteration  890 : error is  0.39707183315
Iteration  900 : error is  0.394868523324
Iteration  910 : error is  0.39300898749
Iteration  920 : error is  0.391310514681
Iteration  930 : error is  0.38962202584
Iteration  940 : error is  0.387835007602
Iteration  950 : error is  0.385823953128
Iteration  960 : error is  0.383686435093
Iteration  970 : error is  0.381603064723
```

```
Iteration  980 : error is  0.379563084041
Iteration  990 : error is  0.377491963716
Iteration  1000 : error is  0.375316062666
```

```
In [297]: plt.figure()

          cmap = plt.get_cmap('Accent', len(unique_labels))

          plt.scatter(Y[:,0], Y[:,1], 20, c=labels, label=labels, cmap=cmap,
                      vmin=min(unique_labels_values), vmax=max(unique_labels_values
          cbar = plt.colorbar(ticks=unique_labels_values)
          cbar.ax.set_yticklabels(unique_labels)  # horizontal colorbar

          plt.show()
```




```
In [116]: unique_labels_values

Out[116]: [1, 2, 3, 5, 4]

In [298]: for key, value in labels_to_int.items():
              print(key, value)

full 1
shuffled 2
mixed 3
```

```
bigbar3 5
bigbar2 4
```

In [ ]: