CSIC5100 HW1 Quesion 1 & 2

LEE, Jooran (SID: 20538819)

Question 1.

In [34]:
```python
import numpy as np
import numpy.linalg as alg
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import matplotlib.cm as cm
from matplotlib.ticker import MaxNLocator
from sklearn.decomposition import PCA
```

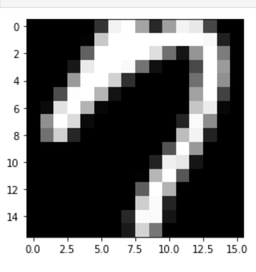
(a) Setting up data matrix

In [35]:
```python
data = pd.read_csv('/Users/jooranlee/Dropbox/Mac/Desktop/CSIC5100/HW1/train.txt')
X = np.array(data, dtype = 'float32')
X = np.transpose(X) # transpose to set up the matrix in pxn
X.shape
```

Out[35]:
```
(256, 644)
```

(b) Compute the sample mean

In [36]:
```python
mu = np.mean(X, axis=0)
n = X.shape[1]
I = np.identity(n)
H = I - np.ones((n, n))/n
X_row = X.dot(H)
X_row = np.transpose(X_row)
X_row_1 = np.reshape(X_row[0,:],(16,16))
imgshow = plt.imshow(X_row_1,cmap='gray')  # plot one of X_tilda

X = np.transpose(X)
X_1 = np.reshape(X[0,:],(16,16))
imgshow = plt.imshow(X_1,cmap='gray') # plot one of X for comparison
```



(c) Calculate SVD

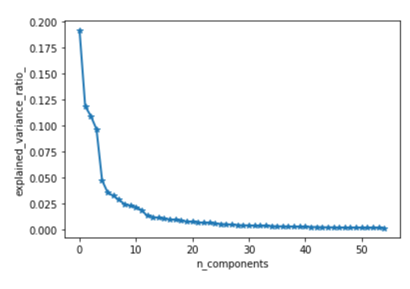
In [37]:
```python
pca = PCA(n_components=100, svd_solver='arpack')
pca.fit(X)
lambda_hat = pca.explained_variance_ratio_
sum = 0
for i in range(len(lambda_hat)):
    sum += lambda_hat[i]
    if sum >= 0.95:
        break
print("k: ", i)
print(sum)
lambda_hat = lambda_hat[:i+1]
print(lambda_hat)
```

```
k:  54
0.950538068311289
[0.19170901 0.11865437 0.10925464 0.09660694 0.04727705 0.03545397
 0.03241991 0.02850023 0.02441354 0.02290041 0.02157202 0.01872944
 0.01312474 0.01203694 0.01143348 0.010344   0.00981531 0.00937494
 0.00859567 0.00774198 0.00746575 0.007072   0.0065004  0.00641337
 0.00599196 0.00533758 0.00519231 0.0047779  0.00437474 0.00410675
 0.00402912 0.00383348 0.00361316 0.00358362 0.00339337 0.00313996
 0.0031172  0.00303619 0.00283155 0.00269428 0.00263524 0.00246222
 0.00245895 0.0022744  0.00215547 0.002139   0.00199543 0.00194769
 0.00193037 0.0019094  0.00178056 0.00170583 0.00163658 0.00154116
 0.00150249]
```

(d) Plot the eigenvalue curve

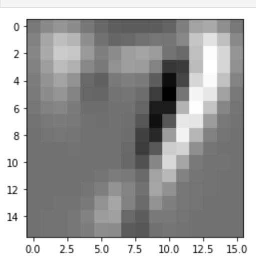
In [38]:
```python
plt.plot(lambda_hat, "*-", linewidth=2)
plt.axis('tight')
plt.xlabel('n_components')
plt.ylabel('explained_variance_ratio_')
```

Out[38]:
```
Text(0, 0.5, 'explained_variance_ratio_')
```

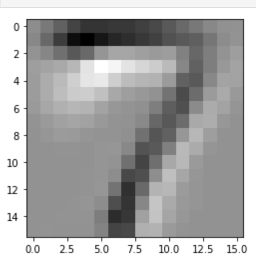


(e) Visualize the top 20 left singular vector

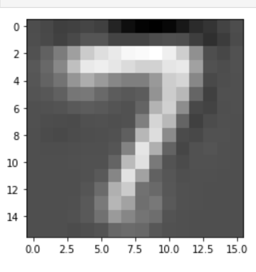
In [39]:
```python
mean_ = pca.mean_
img_mean = np.reshape(mean_,(16,16))
imgshow = plt.imshow(img_mean, cmap='gray')

#top1
c_1 = pca.components_[0]
img_c = np.reshape(c_1,(16,16))
imgshow = plt.imshow(img_c, cmap='gray')
```



In [40]:
```python
#top2
c_2 = pca.components_[1]
img_c = np.reshape(c_2,(16,16))
imgshow = plt.imshow(img_c, cmap='gray')
```



In [41]:
```python
#top3
c_3 = pca.components_[2]
img_c = np.reshape(c_3,(16,16))
imgshow = plt.imshow(img_c, cmap='gray')
```



(f) Ordering of the imanges when k =1

In [42]:
```python
# v_1
mean_projection = np.dot(pca.components_[0], mean_)
v1_list = []
for i in range(X.shape[0]):
    v_1 = (np.dot(pca.components_[0], (X[i, :]) - mean_))
    v1_list.append((i, v_1))
v_1_sorted = sorted(v1_list, key = lambda v: v[1])
print(v_1_sorted)
```

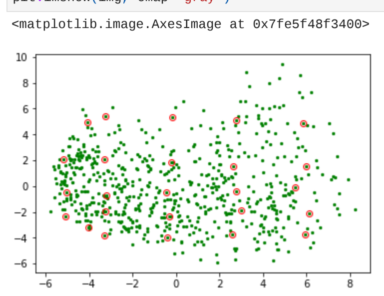
```
[(376, -5.7659492), (279, -5.62024), (209, -5.6015787), (440, -5.5716467), (384, -5.559865), (265, -5.4918246), (562, -5.4162292), (322, -5.4042807), (235, -5.358263), (261, -5.3512588), (151, -5.3475895), (20,
-5.31691), (75, -5.3135185), (426, -5.3056064), (118, -5.2878284), (86, -5.284148), (321, -5.2319155), (610, -5.227692), (563, -5.2197404), (462, -5.1852036), (148, -5.1777096), (202, -5.1697297), (434, -5.14662
17), (574, -5.137332), (278, -5.127184), (478, -5.114633), (561, -5.088279), (590, -5.0850544), (617, -5.0743737), (108, -5.0689626), (289, -5.0529923), (373, -5.0515804), (205, -5.0478864), (400, -5.0457006),
(159, -5.033297), (169, -4.9452934), (155, -4.9438143), (488, -4.9204807), (541, -4.9057403), (172, -4.8942537), (485, -4.880453), (631, -4.851296), (26, -4.848444), (2, -4.844345), (360, -4.843701), (520, -4.84
30777), (58, -4.8064103), (608, -4.7881346), (396, -4.78592), (367, -4.782215), (347, -4.7511), (356, -4.7492695), (54, -4.732635), (204, -4.720461), (355, -4.71956), (366, -4.677823), (293, -4.6721582), (498, -
4.6545177), (365, -4.6467695), (161, -4.629586), (611, -4.625413), (449, -4.5914793), (276, -4.591266), (472, -4.5900636), (70, -4.576172), (550, -4.5713463), (414, -4.543414), (615, -4.5425563), (579, -4.51736
3), (122, -4.5091105), (442, -4.4931097), (487, -4.4828606), (589, -4.479229), (49, -4.457966), (140, -4.4539027), (639, -4.433031), (379, -4.407083), (476, -4.4045267), (191, -4.3942795), (178, -4.37751), (407,
-4.3687773), (25, -4.3644), (578, -4.3332844), (98, -4.331321), (463, -4.3276024), (326, -4.3178263), (586, -4.290318), (263, -4.282902), (117, -4.260739), (227, -4.254371), (484, -4.248514), (34, -4.2423563),
(37, -4.227482), (111, -4.197852), (332, -4.183259), (96, -4.1611843), (110, -4.1605234), (164, -4.158858), (55, -4.1423407), (316, -4.140803), (21, -4.1289625), (410, -4.1277685), (52, -4.1239142), (23, -4.0829
535), (354, -4.079895), (585, -4.0763445), (312, -4.064514), (14, -4.0336676), (69, -4.023127), (253, -4.0116544), (482, -3.9940634), (368, -3.992282), (320, -3.9773088), (595, -3.9742608), (636, -3.9693532), (2
44, -3.957969), (65, -3.942418), (1, -3.907944), (477, -3.9037423), (245, -3.8923697), (454, -3.8682475), (419, -3.861044), (177, -3.849385), (90, -3.83198), (551, -3.8224936), (275, -3.8015304), (468, -3.782031
8), (231, -3.7763155), (4, -3.769145), (524, -3.7586439), (5, -3.7558155), (248, -3.722704), (41, -3.719307), (324, -3.6745324), (228, -3.6537213), (149, -3.642941), (62, -3.6190557), (334, -3.606653), (486, -3.
6037126), (333, -3.601909), (271, -3.5822392), (97, -3.5814419), (362, -3.5500197), (546, -3.5375876), (6, -3.423753), (406, -3.386238), (287, -3.3710794), (530, -3.3458037), (511, -3.3338304), (531, -3.332141
2), (250, -3.3202932), (451, -3.304431), (408, -3.299643), (393, -3.2874613), (247, -3.276475), (437, -3.2730908), (622, -3.273067), (643, -3.272975), (635, -3.2707481), (601, -3.2587805), (576, -3.233945), (33
7, -3.212442), (416, -3.1916327), (68, -3.1631358), (535, -3.1411986), (560, -3.104667), (119, -3.1026921), (80, -3.0937767), (351, -3.054595), (266, -3.0487685), (633, -3.0088992), (350, -3.0053854), (418, -3.0
038347), (61, -2.9433014), (256, -2.9243891), (39, -2.9137313), (584, -2.9118552), (168, -2.841509), (515, -2.8166056), (139, -2.815133), (331, -2.8148346), (411, -2.8088448), (301, -2.792071), (59, -2.7782753),
(158, -2.7638779), (27, -2.743191), (60, -2.698386), (536, -2.6568418), (510, -2.6535313), (349, -2.6515014), (341, -2.6508074), (104, -2.6495984), (211, -2.648203), (113, -2.6438942), (374, -2.6004753), (496, -
2.5821385), (160, -2.5715237), (311, -2.541769), (330, -2.520202), (479, -2.5181794), (504, -2.5034192), (588, -2.4955661), (208, -2.4881186), (174, -2.4844632), (567, -2.4650679), (403, -2.4516196), (162, -2.42
77027), (133, -2.4120317), (623, -2.4040842), (286, -2.400577), (236, -2.391187), (22, -2.3801072), (137, -2.3731236), (251, -2.358328), (582, -2.3523636), (234, -2.323443), (38, -2.3197997), (129, -2.3143735),
(564, -2.3093467), (199, -2.305026), (221, -2.3006363), (390, -2.2830334), (518, -2.2288775), (630, -2.2102604), (465, -2.1808505), (165, -2.1729581), (401, -2.1671052), (56, -2.1317468), (385, -2.0998995), (25
2, -2.0587833), (63, -2.0511208), (383, -2.0204628), (246, -2.0190892), (184, -2.00996), (336, -2.0006921), (489, -1.9910691), (621, -1.971701), (386, -1.8577068), (397, -1.8529617), (543, -1.8407071), (260, -1.
8376958), (469, -1.8251705), (120, -1.8165884), (480, -1.76439), (620, -1.7597556), (409, -1.7512329), (525, -1.7179615), (339, -1.715154), (269, -1.7074199), (352, -1.7031193), (225, -1.6929474), (109, -1.68905
5), (363, -1.6859845), (503, -1.6734335), (144, -1.6571372), (604, -1.6510541), (73, -1.5763756), (315, -1.5721505), (267, -1.5348012), (156, -1.4895146), (239, -1.4702415), (28, -1.4431376), (447, -1.3900765),
(466, -1.3799655), (313, -1.3583243), (142, -1.3427625), (121, -1.3357124), (600, -1.3292848), (446, -1.3164178), (220, -1.3117632), (319, -1.2984141), (427, -1.2896274), (581, -1.2827933), (516, -1.254932), (23
3, -1.2391226), (190, -1.2290967), (306, -1.2279482), (273, -1.185682), (372, -1.1780436), (394, -1.156592), (494, -1.1430984), (257, -1.1320742), (415, -1.0949405), (83, -1.0793095), (145, -1.0688261), (457, -
1.0347351), (300, -1.0163378), (131, -0.9534104), (519, -0.9036411), (558, -0.8903172), (497, -0.88462067), (229, -0.87791276), (116, -0.8674176), (232, -0.7968311), (126, -0.7782932), (641, -0.77816844), (186,
-0.73398644), (138, -0.66695774), (340, -0.6580924), (44, -0.6370044), (71, -0.60794586), (77, -0.600726), (79, -0.5752711), (180, -0.55601573), (507, -0.555673), (592, -0.54363084), (612, -0.54236364), (24, -0.
5276709), (493, -0.520534), (219, -0.5100746), (268, -0.443007), (32, -0.43716937), (529, -0.42056262), (506, -0.41804403), (378, -0.40727055), (532, -0.3588668), (603, -0.34998325), (17, -0.29718527), (572, -0.
29692078), (224, -0.2889006), (348, -0.27801937), (167, -0.2776247), (557, -0.24942613), (29, -0.22337236), (452, -0.21896306), (243, -0.18899745), (613, -0.18545112), (40, -0.15700865), (237, -0.14733449), (15
3, -0.13097277), (544, -0.12799948), (619, -0.124260515), (580, -0.11734554), (31, -0.09970337), (3, -0.09565422), (242, -0.0742936), (417, -0.07182335), (540, -0.05557047), (597, -0.045818828), (483, -0.0401301
1), (490, 0.0078053325), (526, 0.01478947), (323, 0.03979391), (593, 0.042011783), (114, 0.055007815), (357, 0.0887121), (522, 0.104175374), (262, 0.11248952), (568, 0.12548344), (382, 0.14521524), (638, 0.17767
164), (210, 0.20915377), (375, 0.23706745), (545, 0.241393), (74, 0.24357927), (280, 0.27631176), (8, 0.2951115), (238, 0.35523918), (106, 0.36069742), (46, 0.4571926), (616, 0.5375049), (132, 0.54057395), (296,
0.5536691), (570, 0.5728945), (342, 0.5946933), (9, 0.60655254), (453, 0.61916363), (134, 0.64773446), (183, 0.68863857), (282, 0.70953436), (154, 0.7199193), (443, 0.7940901), (82, 0.814558), (87, 0.836367), (4
05, 0.8561605), (288, 0.8561766), (302, 0.8629788), (325, 0.873052), (136, 0.8895217), (99, 0.89625067), (123, 0.9344305), (495, 0.97568417), (143, 0.99248743), (571, 1.0045273), (387, 1.0429606), (381, 1.048834
8), (512, 1.0644313), (421, 1.0788932), (217, 1.113449), (173, 1.1168225), (179, 1.1282548), (78, 1.154752), (594, 1.1625332), (230, 1.2122265), (259, 1.2549751), (308, 1.2647139), (274, 1.2898746), (329, 1.2936
122), (12, 1.3321831), (7, 1.3589063), (353, 1.3602389), (112, 1.3792928), (566, 1.3854221), (391, 1.4409342), (103, 1.4573615), (436, 1.468966), (628, 1.5077167), (399, 1.5560555), (537, 1.5783541), (458, 1.591
5645), (193, 1.5941389), (596, 1.6352866), (283, 1.6529391), (637, 1.6608641), (627, 1.6737462), (53, 1.6814944), (157, 1.7001499), (471, 1.736767), (555, 1.7449274), (439, 1.7930396), (642, 1.8363755), (508, 1.
840263), (361, 1.847708), (299, 1.8498489), (290, 1.8510098), (549, 1.8592429), (11, 1.8868892), (203, 1.8889192), (425, 1.889991), (303, 1.9217548), (500, 1.9543228), (214, 1.9571365), (389, 1.9684818), (91, 1.
9812893), (317, 2.0064983), (226, 2.0078702), (307, 2.0445716), (634, 2.0596402), (517, 2.090681), (264, 2.1043277), (310, 2.1093352), (583, 2.1146362), (420, 2.1256423), (556, 2.1555603), (505, 2.168726), (345,
2.1868045), (222, 2.214088), (292, 2.236528), (467, 2.2491312), (76, 2.2595696), (502, 2.2980795), (66, 2.307055), (272, 2.3266249), (16, 2.3697233), (402, 2.3756294), (388, 2.3883412), (30, 2.4007232), (166, 2.
4197476), (294, 2.4505954), (196, 2.455892), (632, 2.4793837), (542, 2.4840274), (270, 2.502842), (95, 2.5277388), (459, 2.5325665), (100, 2.5446136), (456, 2.5681825), (255, 2.5818417), (377, 2.6017208), (206,
2.616119), (575, 2.617332), (277, 2.622541), (428, 2.6730587), (88, 2.7026758), (528, 2.7078278), (591, 2.7141871), (346, 2.7192216), (435, 2.7270477), (547, 2.758829), (175, 2.7642124), (249, 2.7861543), (614,
2.8579183), (223, 2.8695092), (626, 2.877091), (533, 2.9072886), (42, 2.9201646), (413, 2.9622781), (36, 2.996934), (72, 3.0349817), (213, 3.155935), (48, 3.17422), (338, 3.1818604), (640, 3.2160368), (599, 3.23
25206), (189, 3.2481022), (625, 3.2826602), (43, 3.3047674), (185, 3.3094637), (474, 3.3197095), (473, 3.3588672), (605, 3.361715), (422, 3.460567), (359, 3.474064), (559, 3.5380309), (92, 3.5524745), (47, 3.611
5327), (521, 3.625118), (125, 3.649173), (327, 3.6761367), (398, 3.70244), (258, 3.7052073), (475, 3.709403), (460, 3.7463634), (298, 3.7569335), (513, 3.7980318), (187, 3.7991028), (412, 3.8046072), (85, 3.8183
53), (430, 3.869589), (135, 3.887476), (523, 3.90596), (170, 3.9063811), (569, 3.908512), (212, 3.9737318), (431, 3.984134), (0, 3.984871), (553, 3.9903293), (501, 3.9996197), (554, 4.0010257), (328, 4.0198455),
(147, 4.0567074), (33, 4.0684023), (424, 4.0867805), (461, 4.130041), (618, 4.1321034), (433, 4.177686), (577, 4.2195973), (254, 4.235586), (45, 4.2464647), (335, 4.2606883), (241, 4.2760425), (455, 4.298154),
(13, 4.302039), (629, 4.306821), (380, 4.3218927), (128, 4.347275), (492, 4.349016), (552, 4.361728), (218, 4.3687954), (441, 4.3706856), (318, 4.3800273), (607, 4.4426537), (404, 4.4440722), (432, 4.454863), (2
84, 4.4562488), (514, 4.460986), (395, 4.524272), (344, 4.5331464), (314, 4.543898), (18, 4.5467954), (448, 4.582946), (491, 4.5965676), (481, 4.6103005), (124, 4.611637), (392, 4.620283), (499, 4.6219773), (14
6, 4.6257734), (67, 4.6511097), (102, 4.6566963), (94, 4.679815), (182, 4.729253), (534, 4.7576075), (609, 4.767764), (130, 4.781751), (539, 4.80536), (81, 4.851967), (89, 4.864304), (35, 4.8710036), (93, 4.9299
784), (195, 4.937922), (602, 4.9589796), (464, 4.9987717), (192, 5.0042477), (538, 5.0319633), (548, 5.0349617), (57, 5.0431957), (194, 5.1026964), (509, 5.1457434), (200, 5.149559), (105, 5.1862555), (281, 5.18
73646), (84, 5.2000318), (309, 5.2568736), (364, 5.28148), (438, 5.2973604), (573, 5.3426104), (101, 5.3767986), (470, 5.394759), (450, 5.451957), (127, 5.4727874), (444, 5.4811945), (216, 5.496208), (423, 5.527
686), (107, 5.54103), (370, 5.5644135), (201, 5.5718575), (176, 5.6041665), (598, 5.680011), (188, 5.700238), (10, 5.773925), (606, 5.825446), (141, 5.831231), (115, 5.854927), (215, 5.932913), (171, 5.9440727),
(295, 6.009368), (50, 6.030156), (163, 6.0473347), (291, 6.1051426), (565, 6.1186295), (429, 6.1664696), (51, 6.2025146), (198, 6.2831993), (358, 6.2953115), (197, 6.32199), (152, 6.383405), (527, 6.4386816), (2
40, 6.474352), (297, 6.489066), (369, 6.5015545), (15, 6.642235), (624, 6.707614), (445, 7.045072), (64, 7.133168), (181, 7.2333193), (587, 7.2449675), (305, 7.290532), (371, 7.3478227), (285, 7.4061203), (207,
7.438265), (343, 7.5009127), (304, 7.52311), (150, 7.941571), (19, 8.224452)]
```

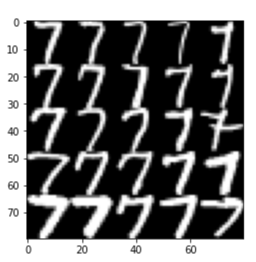
(g) Plotting the scatter plot when k = 2

In [43]:
```python
# v_2
mean_projection = np.dot(pca.components_[1], mean_)
v2_list = []
for i in range(X.shape[0]):
    v_2 = (np.dot(pca.components_[1], (X[i, :]) - mean_ - v1_list[i][1]*pca.components_[0]))
    v2_list.append((i, v_2))
```

In [25]:
```python
points = []
for i in range(X.shape[0]):
    points.append([v1_list[i][1], v2_list[i][1]])
points = np.array(points)
# quantile points
v1_percentile = np.quantile(points[:, 0], [0.05, 0.25, 0.50, 0.75, 0.95], interpolation='lower')
v2_percentile = np.quantile(points[:, 1], [0.05, 0.25, 0.50, 0.75, 0.95], interpolation='lower')
v1_percentile_index = []
v2_percentile_index = []

# scatter plot for projection
plt.figure()
plt.scatter(points[:, 0], points[:, 1], c='g', s=5)
img = np.zeros([16*5, 16*5])
for i in range(5):
    for j in range(5):
        point = [v1_percentile[i], v2_percentile[j]]
        point_index = (np.sum((points[:, :] - point)**2, axis=1)).argmin()
        plt.plot(points[point_index, 0], points[point_index, 1], 'o', markerfacecolor='none', c='r')

        # corresponding images
        img[16*i:16*(i+1), 16*j:16*(j+1)] = np.reshape(X[point_index,:],(16,16))

# plot corresponding images
plt.figure()
plt.imshow(img, cmap='gray')
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x7fe5f48f3400>
```

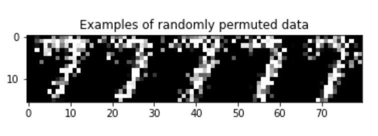


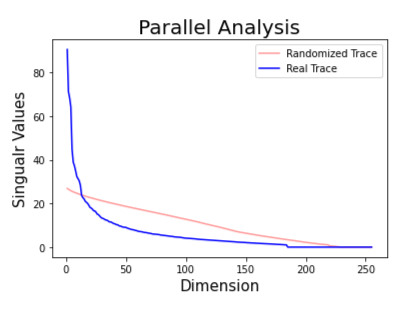


(h) parallel analyis with permutation test

In [27]:
```python
# permutation
X_permute = np.copy(X)
m = X.shape[1]

pca = PCA(n_components=m-1, svd_solver='arpack')
pca.fit(X)
lambda_hat = pca.explained_variance_ratio_

sum_eigens = np.empty(X.shape[1]-1)
# Repeat 100 times
for r in range(100):
    # Permutation
    for i in range(m):
        X_permute[:, i] = np.random.permutation(X_permute[:, i])
    pca_random = PCA(n_components=m-1, svd_solver='arpack')
    pca_random.fit(X_permute)
    sum_eigens = sum_eigens + pca_random.singular_values_

    if r == 0:
        img = np.zeros([16, 16*5])
        for i in range(5):
            img[:, 16*i:16*(i+1)] = np.reshape(X_permute[i, :],(16,16))
        plt.imshow(img,cmap='gray')
        plt.title("Examples of randomly permuted data")

# Average over the number of runs
avg_eigens = sum_eigens / 100

plt.figure()
plt.plot(range(1, m), avg_eigens, 'r', label='Randomized Trace', alpha=0.4)
plt.plot(range(1, m), pca.singular_values_, 'b', label='Real Trace')

plt.title('Parallel Analysis', {'fontsize': 20})
plt.xlabel('Dimension', {'fontsize': 15})
# plt.xticks(ticks=range(1, m+1), labels=range(1, m+1))
plt.ylabel('Singualr Values', {'fontsize': 15})
plt.legend()
```

Out[27]:
```
<matplotlib.legend.Legend at 0x7fe5f40216d0>
```





Question 2.

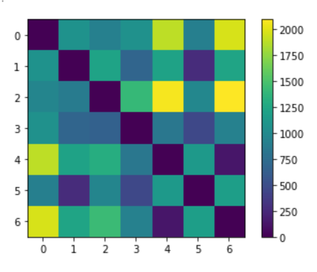
(a) Collect the pairwise air traveling distances

In [117…
```python
dist = {'beijing':{'beijing':0, 'shanghai':1070.48, 'seoul':956.30, 'wuhan':1057.09, 'guangzhou':1890.71, 'nanjing':902.02, 'hongkong':1973.70},
        'shanghai':{'beijing':1070.48, 'shanghai':0, 'seoul':870.86, 'wuhan':687.30, 'guangzhou':1206.63, 'nanjing':268.02, 'hongkong':1227.83},
        'seoul':{'beijing':913.89, 'shanghai':1218.90, 'seoul':0, 'wuhan':648.95, 'guangzhou':1309.88, 'nanjing':950.98, 'hongkong':1428.01},
        'wuhan':{'beijing':1057.09, 'shanghai':687.30, 'seoul':1404.38, 'wuhan':0, 'guangzhou':834.25, 'nanjing':458.49, 'hongkong':922.97},
        'guangzhou':{'beijing':1890.71, 'shanghai':1206.63, 'seoul':2070.53, 'wuhan':834.25, 'guangzhou':0, 'nanjing':1129.45, 'hongkong':129.70},
        'nanjing':{'beijing':902.02, 'shanghai':268.02, 'seoul':968.51, 'wuhan':458.49, 'guangzhou':1129.45, 'nanjing':0, 'hongkong':1178.23},
        'hongkong':{'beijing':1973.70, 'shanghai':1227.83, 'seoul':2097.78, 'wuhan':922.97, 'guangzhou':129.70, 'nanjing':1178.23, 'hongkong':0}}
```

In [118…
```python
df = pd.DataFrame(dist)
print(df)
D = df.values
plt.imshow(D)
plt.colorbar()
```

```
           beijing  shanghai    seoul    wuhan  guangzhou  nanjing  hongkong
beijing       0.00   1070.48   913.89  1057.09    1890.71   902.02   1973.70
shanghai   1070.48      0.00  1218.90   687.30    1206.63   268.02   1227.83
seoul       956.30    870.86     0.00  1404.38    2070.53   968.51   2097.78
wuhan      1057.09    687.30   648.95     0.00     834.25   458.49    922.97
guangzhou  1890.71   1206.63  1309.88   834.25       0.00  1129.45    129.70
nanjing     902.02    268.02   950.98   458.49    1129.45     0.00   1178.23
hongkong   1973.70   1227.83  1428.01   922.97     129.70  1178.23      0.00
```

Out[118]:
```
<matplotlib.colorbar.Colorbar at 0x7fe5f6db6790>
```



In [119…
```python
print(D)
```

```
[[   0.   1070.48  913.89 1057.09 1890.71  902.02 1973.7 ]
 [1070.48    0.   1218.9   687.3  1206.63  268.02 1227.83]
 [ 956.3   870.86    0.   1404.38 2070.53  968.51 2097.78]
 [1057.09  687.3   648.95    0.    834.25  458.49  922.97]
 [1890.71 1206.63 1309.88  834.25    0.   1129.45  129.7 ]
 [ 902.02  268.02  950.98  458.49 1129.45    0.   1178.23]
 [1973.7  1227.83 1428.01  922.97  129.7  1178.23    0.  ]]
```

(b) Codes of multidimensional scaling algorithm for D

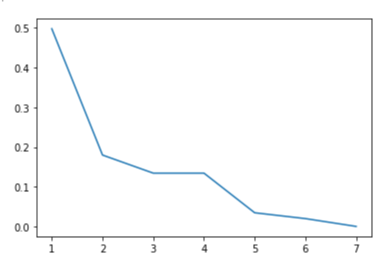
In [120…
```python
m = np.ones(shape=(len(D),1))*(1/len(D))
C = np.identity(n=len(m)) - np.full(shape=(len(m),len(m)),fill_value=m)
S = -0.5*np.matmul(C,np.matmul(D,np.transpose(C)))
eigen = np.linalg.eig(S)
U = eigen[1]
```

(c) Plot the normalized eigenvalues in a descending order

In [121…
```python
eigen = sorted(np.array(eigen[0]), reverse=True)
sum_eigen = np.sum(eigen)
array = np.array(eigen/sum_eigen)
print('Eigenvalues are: ', np.real(array))
plt.plot(range(1, 8), array)
```

```
Eigenvalues are:  [ 4.97821786e-01  1.79726434e-01  1.34185937e-01  1.34185937e-01
  3.43940601e-02  1.96858463e-02 -2.72587411e-17]
```

Out[121]:
```
[<matplotlib.lines.Line2D at 0x7fe5f7032550>]
```

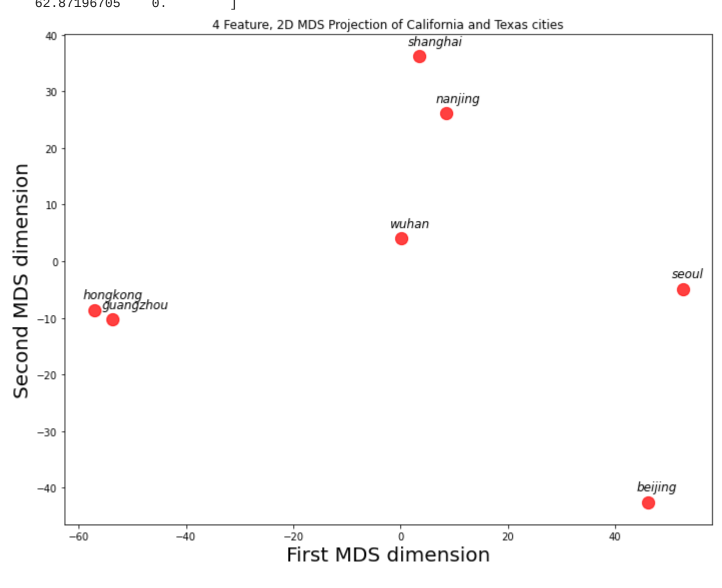


There is no negative eigenvalues. The last -2.72587411e-17 value indicates an extremely small negative number, therefore cannot be viewed as a negative number.

(d) Make a scatter plot of cities

In [122…
```python
eigen = np.array(eigen)
Lambda = np.where(eigen<=0, 0, eigen)

print('Eigenvalues are:\n',np.real(Lambda))

scores = np.matmul(np.sqrt(np.linalg.inv(np.diag(m[:,0]))),np.matmul(U,np.sqrt(np.diag(Lambda))))
x_proj = scores[:,0]
y_proj = scores[:,1]

fig1 = plt.figure()
plt.scatter(x_proj[:],y_proj[:], color='red', s=150, alpha=0.75)

for i, txt in enumerate(df.iteritems()):
    plt.annotate(txt[0], (x_proj[i]-2, y_proj[i]+2), size=12, style='italic')

plt.title('4 Feature, 2D MDS Projection of California and Texas cities')
plt.xlabel('First MDS dimension',size=20)
plt.ylabel('Second MDS dimension',size=20)
plt.subplots_adjust(left=0.0, bottom=0.0, right=1.5, top=1.7, wspace=0.5, hspace=0.3)
plt.show()
```

```
Eigenvalues are:
 [1589.92579567  574.0039931   428.55834941  428.55834941  109.84654536
   62.87196705    0.        ]
```



The graph reflects the position relationshi: Seoul, and shanghai were the east and north of most cities, respectively. After the projection, the relative position actually remains unchanged.

3. *Positive Semi-definiteness:* Recall that a n-by-n real symmetric matrix K is called positive semi-definite (*p.s.d.* or $K \succeq 0$) iff for every $x \in \mathbb{R}^n$, $x^T K x \geq 0$.

(a) Show that $K \succeq 0$ if and only if its eigenvalues are all nonnegative.

(b) Show that $d_{ij} = K_{ii} + K_{jj} - 2K_{ij}$ is a squared distance function, *i.e.* there exists vectors $u_i, v_j \in \mathbb{R}^n$ $(1 \leq i, j \leq n)$ such that $d_{ij} = \|u_i - u_j\|^2$.

(c) Let $\alpha \in \mathbb{R}^n$ be a signed measure s.t. $\sum_i \alpha_i = 1$ (or $e^T\alpha = 1$) and $H_\alpha = I - e\alpha^T$ be the Householder centering matrix. Show that $B_\alpha = -\frac{1}{2}H_\alpha D H_\alpha^T \succeq 0$ for matrix $D = [d_{ij}]$.

(d) If $A \succeq 0$ and $B \succeq 0$ $(A, B \in \mathbb{R}^{n\times n})$, show that $A + B = [A_{ij} + B_{ij}]_{ij} \succeq 0$ (elementwise sum), and $A \circ B = [A_{ij}B_{ij}]_{ij} \succeq 0$ (Hadamard product or elementwise product).

(a) Given that $K \geq 0 \iff X \in R^n,\ X^T K X \geq 0$.

we will prove that $X \in R^n,\ X^T K X \geq 0 \iff$ all eigenvalues are non-negative.

$\Leftarrow$: $\exists Q$ s.t. $K = Q\Lambda Q^T$

$x^T K x = x^T(Q\Lambda Q^T)x = y^T \Lambda y = \lambda_1 y_1^2 + \cdots + \lambda_n y_n^2 \geq 0$

where $y = Q^T x = (y_1, y_2, \cdots, y_n) \neq 0$.

$\Rightarrow$: K is a real symmetric matrix. $(K = Q\lambda Q^T)$

$\therefore X^T K X \geq 0 \Rightarrow x^T(Q\lambda Q^T) \geq 0$

If $y = Q^T x$, $y\Lambda y^T \geq 0 \quad \therefore \lambda_i \geq 0$.

(b) $d_{ij} = K_{ii} + K_{jj} - 2K_{ij} = u_i^T K u_i + v_j^T K v_j - 2u_i^T K v_j$

$= (u_i - v_j)^T K (u_i - v_j)$

as K is a symmetric matrix $\Rightarrow d_{ij}$ is a squared distance function.

(c) squared distance function is conditionally negative define (c.n.d).

we will prove that $B_\alpha = -\frac{1}{2}H_\alpha D H_\alpha^T \geq 0$ for $D = [d_{ij}] \iff [d_{ij}]$ is c.n.d.

$\Leftarrow$: $\exists x \in R^n$, $x^T B_\alpha x = -\frac{1}{2} x^T H_\alpha D H_\alpha^T x = -\frac{1}{2}(H_\alpha^T x)^T D (H_\alpha^T x)$

We know that:

$e^T \cdot H_\alpha^T x = e^T(I - e^T)x = (e - e^T\cdot\alpha)\, e^T \cdot x = 0$ (as $e^T\cdot\alpha = 1$ for α)

$\therefore x^T B_\alpha x = \frac{-1}{2}(H_\alpha^T x)^T D (H_\alpha^T x) \geq 0$ as d is c.n.d.

$\Rightarrow$: For $\forall x \in R^n$ s.t. $e^T x = 0$, we have $H a^T x = (I - a e^T) x =$

$x - a e^T x = x.$

$\therefore$ $x^T D x = (H a^T x) D (H a^T x) = x^T H a D H a^T x = -2 x^T B a X \leq 0.$

(d) $A + B = [A_{ij} + B_{ij}]_{ij} \geq 0$ because $A \geq 0$ and $B \geq 0$

(due to property of positive semi-define matrix).

4. *Distance:* Suppose that $d : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is a distance function.

(a) Is d^2 a distance function? Prove or give a counter example.

(b) Is $\sqrt{d}$ a distance function? Prove or give a counter example.

Four characteristics of distance function:

i) $d(A, B) \geq 0$

ii) $d(A, A) = 0$

iii) $d(A, B) = d(B, A)$

iv) triangle inequality: $d(A, B) \leq d(A, C) + d(C, B)$.

$\therefore$ (a) No. The counter example is:

$d(A, B) = 5$ $\quad$ $d(A, B)^2 = 5^2 = 25$

$d(A, C) = 2$ $\quad$ $d(A, C)^2 = 2^2 = 4$

$d(B, C) = 1$ $\quad$ $d(B, C)^2 = 1^2 = 1$

However, $d(A, B)^2 \geq d(A, C)^2 + d(B, C)^2$, which doesn't satisfy the triangle inequality.

(b) Yes. i) $d(A, B) \geq 0 \Rightarrow \sqrt{d(A, B)} \geq 0$

ii) $d(A, A) = 0 \Rightarrow \sqrt{d(A, A)} = 0$

iii) $d(A, B) = d(B, A) \Rightarrow \sqrt{d(A, B)} = \sqrt{d(B, A)}$

IV) $d(A, B) \leq d(B, C) + d(C, B)$

$\Rightarrow d(A, B) \leq d(B, C) + d(C, B) + 2\sqrt{d(A, C) d(C, B)}$

$\Rightarrow (\sqrt{d(A, B)})^2 \leq (\sqrt{d(B, C)} + \sqrt{d(C, B)})^2$

5. **Singular Value Decomposition:* The goal of this exercise is to refresh your memory about the singular value decomposition and matrix norms. A good reference to the singular value decomposition is Chapter 2 in this book:
Matrix Computations, Golub and Van Loan, 3rd edition.

(a) *Existence:* Prove the existence of the singular value decomposition. That is, show that if A is an $m \times n$ real valued matrix, then $A = U\Sigma V^T$, where U is $m \times m$ orthogonal matrix, V is $n \times n$ orthogonal matrix, and $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p)$ (where $p = \min\{m, n\}$) is an $m \times n$ diagonal matrix. It is customary to order the singular values in decreasing order: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$. Determine to what extent the SVD is unique. (See Theorem 2.5.2, page 70 in Golub and Van Loan).

(b) *Best rank-k approximation - operator norm:* Prove that the "best" rank-k approximation of a matrix in the operator norm sense is given by its SVD. That is, if $A = U\Sigma V^T$ is the SVD of A, then $A_k = U\Sigma_k V^T$ (where $\Sigma_k = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0)$ is a diagonal matrix containing the largest k singular values) is a rank-k matrix that satisfies

$$\|A - A_k\| = \min_{\text{rank}(B)=k} \|A - B\|.$$

(Recall that the operator norm of A is $\|A\| = \max_{\|x\|=1} \|Ax\|$. See Theorem 2.5.3 (page 72) in Golub and Van Loan).

(c) *Best rank-k approximation - Frobenius norm:* Show that the SVD also provides the best rank-k approximation for the Frobenius norm, that is, $A_k = U\Sigma_k V^T$ satisfies

$$\|A - A_k\|_F = \min_{\text{rank}(B)=k} \|A - B\|_F.$$

(a) Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ be the unit 2-norm vectors that satisfy $Ax = \sigma y$ with $\sigma = \|A\|_2$. From the Golub and Van Loan Theorem, there exists $V_2 \in \mathbb{R}^{n \times (n-1)}$ and $U_2 \in \mathbb{R}^{m \times (m-1)}$. so, $V = [x_2 V_2] \in \mathbb{R}^{n \times n}$ and $U = [y U_2] \in \mathbb{R}^{m \times m}$ are orthogonal to each other.

$$\because \left\|A_1\left(\begin{bmatrix} \sigma \\ w \end{bmatrix}\right)\right\|_2^2 \geq (\sigma^2 + w^T w)^2$$

and if $w=0$ (as $\sigma^2 = \|A\|_2^2 = \|A_1\|_2^2$), $\underline{U^T A V = \begin{bmatrix} \sigma & w^T \\ 0 & B \end{bmatrix} \equiv A_1}$

(b) As $U^T A V = \text{diag}(\sigma_1, \cdots \sigma_K, 0, \cdots, 0)$, it follows that $\text{rank}(A_K) = K$, and $U^T(A - A_K)V = \text{diag}(0, \cdots, 0, \sigma_{K+1}, \cdots, \sigma_p)$, and therefore

$$\|A - A_K\|_2 = \sigma_{K+1}.$$

And if $\text{rank}(B) = K$ for some $B \in \mathbb{R}^{m \times n}$, we can say $\text{null}(B) = \text{span}(x_1, \cdots, x_{n-K})$ as the orthogonal vectors are $x_1, x_2, \cdots, x_{n-K}$.

And the dimension argument shows that

$$\operatorname{span}\{x_1, \cdots, x_{n-k}\} \cap \operatorname{span}\{v_1, \cdots, v_{k+1}\} \neq \{0\}.$$

Let z be an unit 2-norm vector in this intersection. Since $Bz = 0$, and

$$Az = \sum_{i=1}^{k+1} \sigma_i (v_i^T z) u_i,$$

$$\therefore \quad \|A-B\|_2^2 \geq \|(A-B)z\|_2^2 = \|Az\|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 (v_i^T z)^2 \geq \sigma_{k+1}^2$$

(c) let's say B minimizes $\|A-B\|_F^2$ among all ranks. And let V be the space spanned by the rows of the B, and the highest dimension of B is at k.

If the B minimizes $\|A-B\|_F^2$, it must be that each row of B is the projection of the corresponding row of A onto V.

And $\|A-B\|_F^2$ = sum of the squared distances of rows of A onto B, as each row of B is the projection of the corresponding row of A.

$\therefore$ A_k minimizes the sum of squared distance of rows of A to any k-dimensional space, mathematically:

$$\|A - A_k\|_F \leq \|A - B\|_F.$$