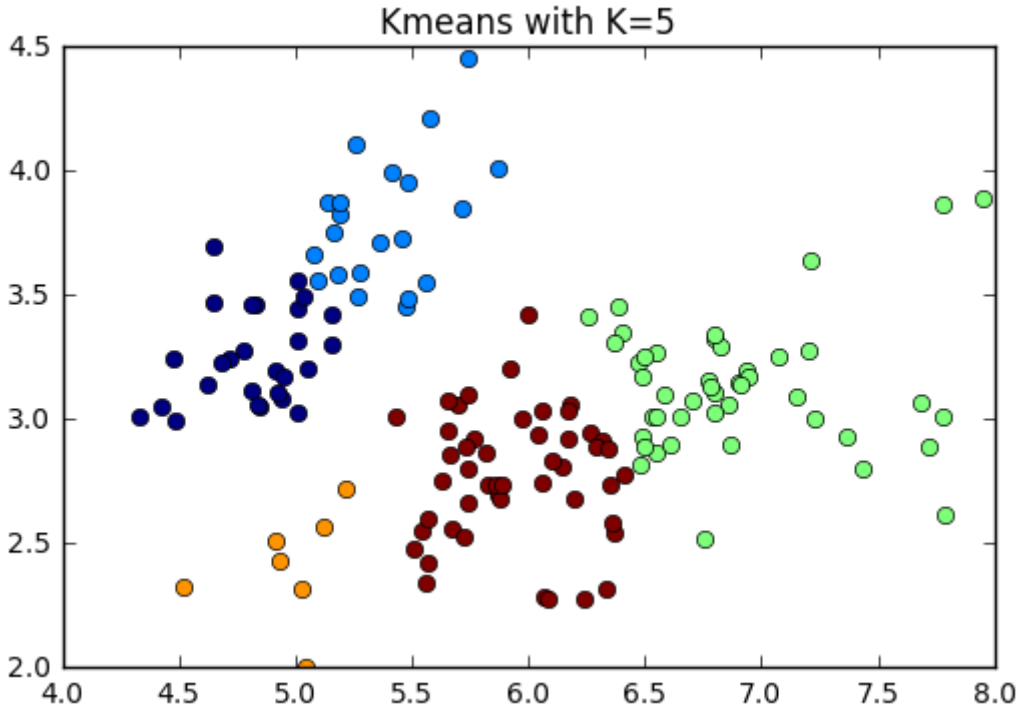
In [83]:

```
#1a

import numpy as np
import math

#math.log?
data = np.genfromtxt("A5/data/iris.txt",delimiter=None)
X=data[:,0:2]
X1 = data[:,0]
X2 = data[:,1]
#print X
```

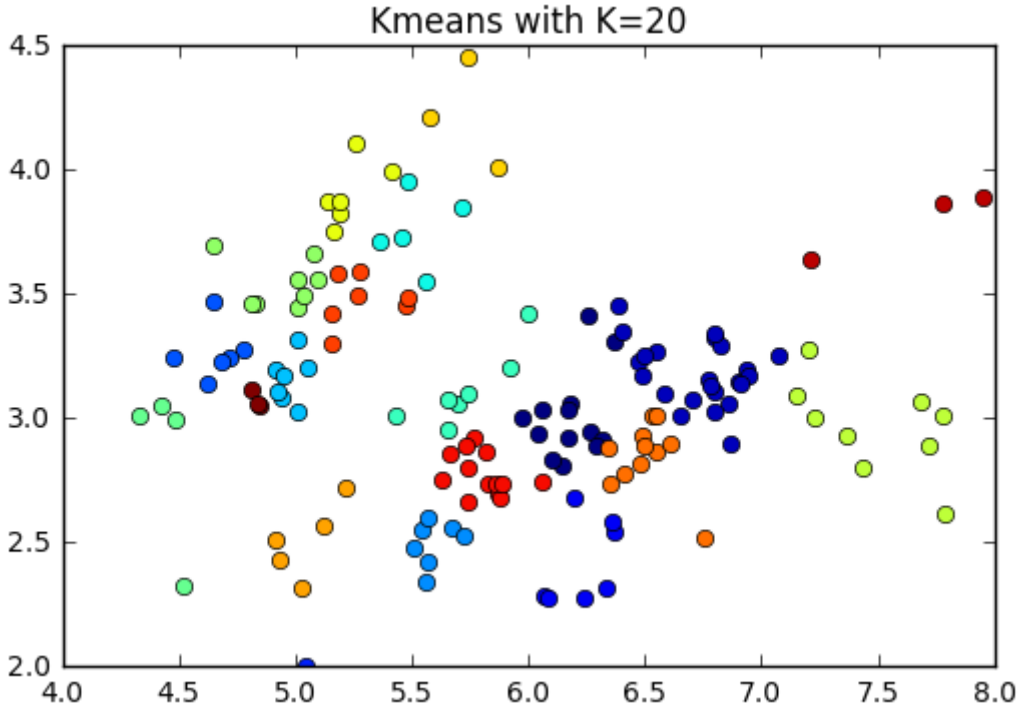
In [86]:

```
#1b

import mltools as ml
import matplotlib.pyplot as plt
K =[5,20]
init_val=['random','farthest','k++']
for m,n in enumerate(init_val):
    for i,j in enumerate(K):
        z,c,sumd= ml.cluster.kmeans(X,j,n)
        plt.clf()
        #print X.shape
        ml.plotClassify2D(None,X,z)
        #plt.xlabel('')
        #plt.ylabel('')
        if j==5:
            plt.title('Kmeans with K=5')
        else:
            plt.title('Kmeans with K=20')
        print "score"+str(sumd)
        plt.show()
```

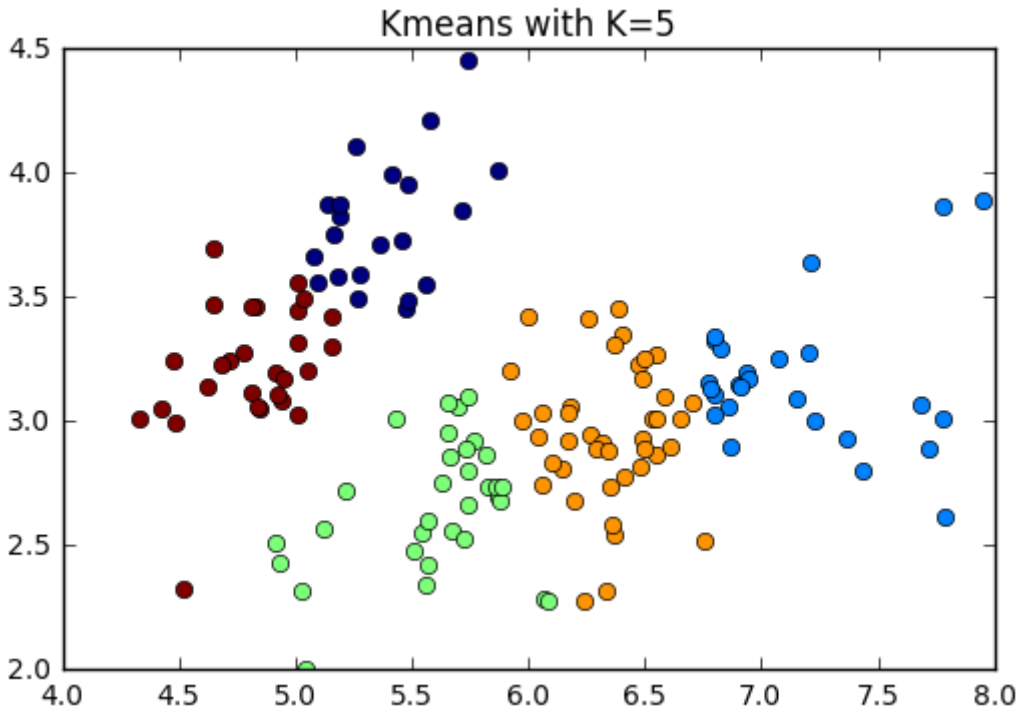
```
score23.8807913892
```



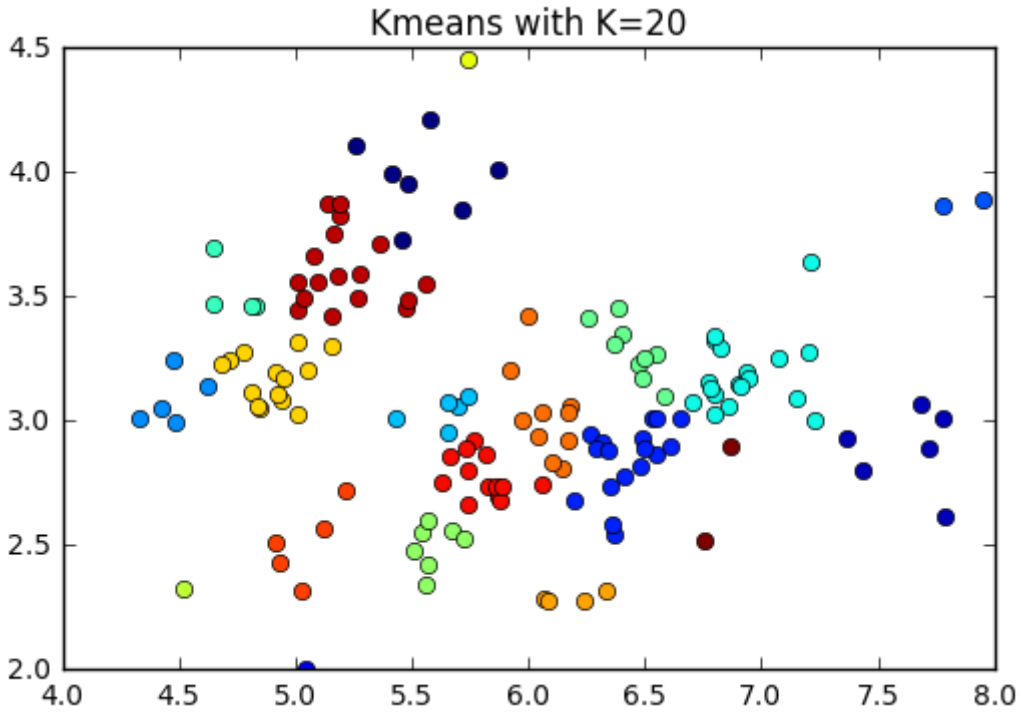

```
score5.73121960826
```



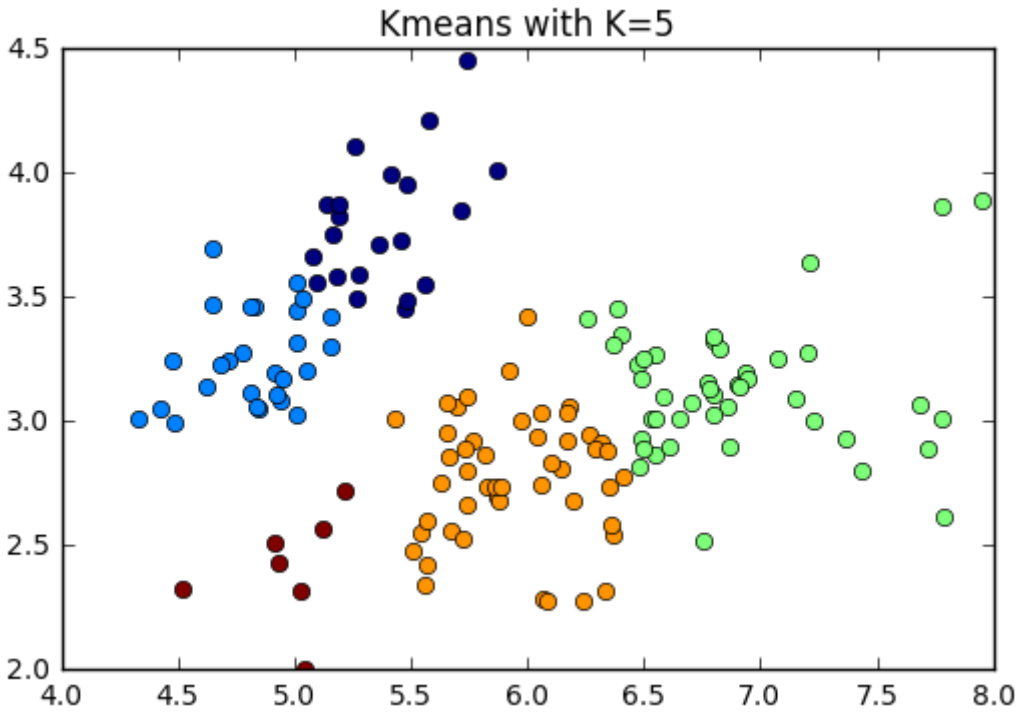

```
score21.341435585
```



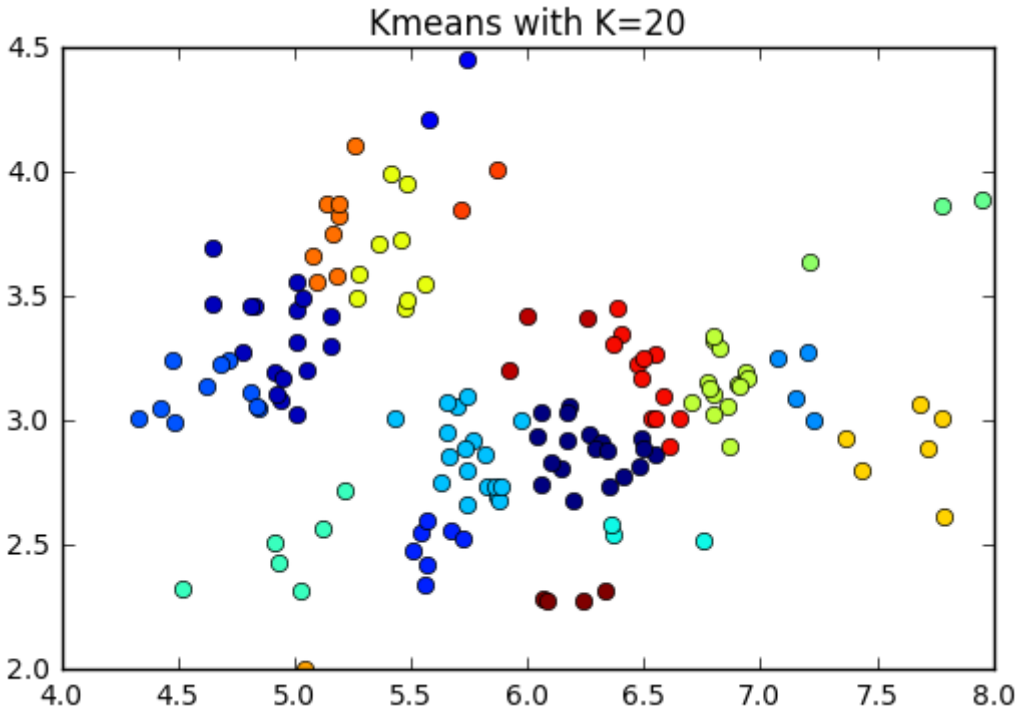

```
score4.70578909628
```




```
score23.8807913892
```



score5.07683016535

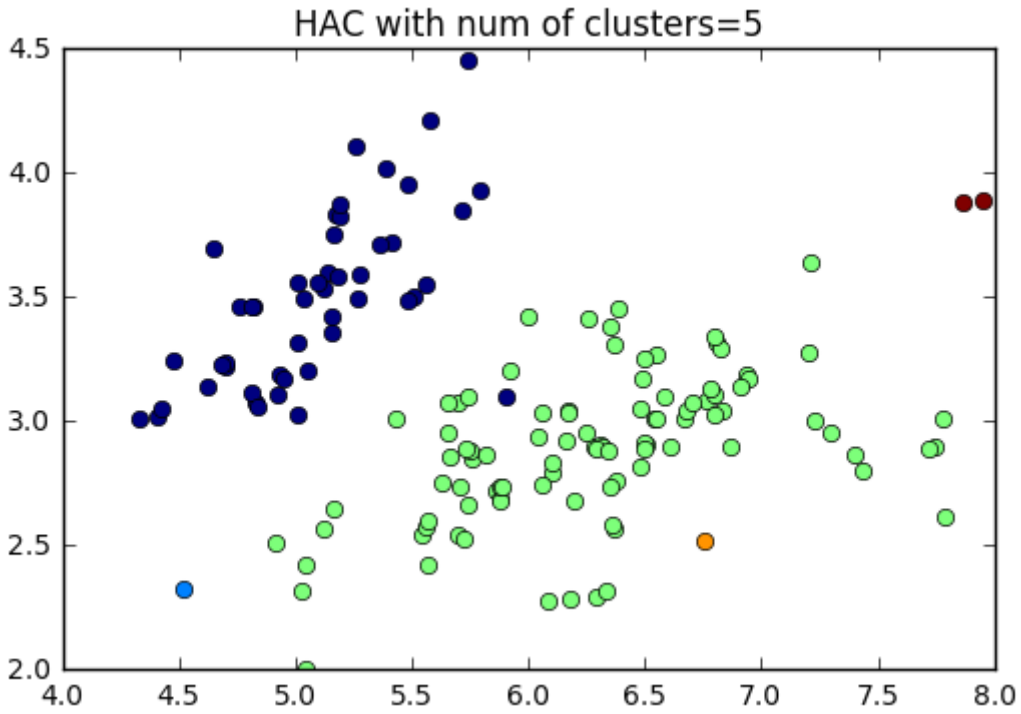


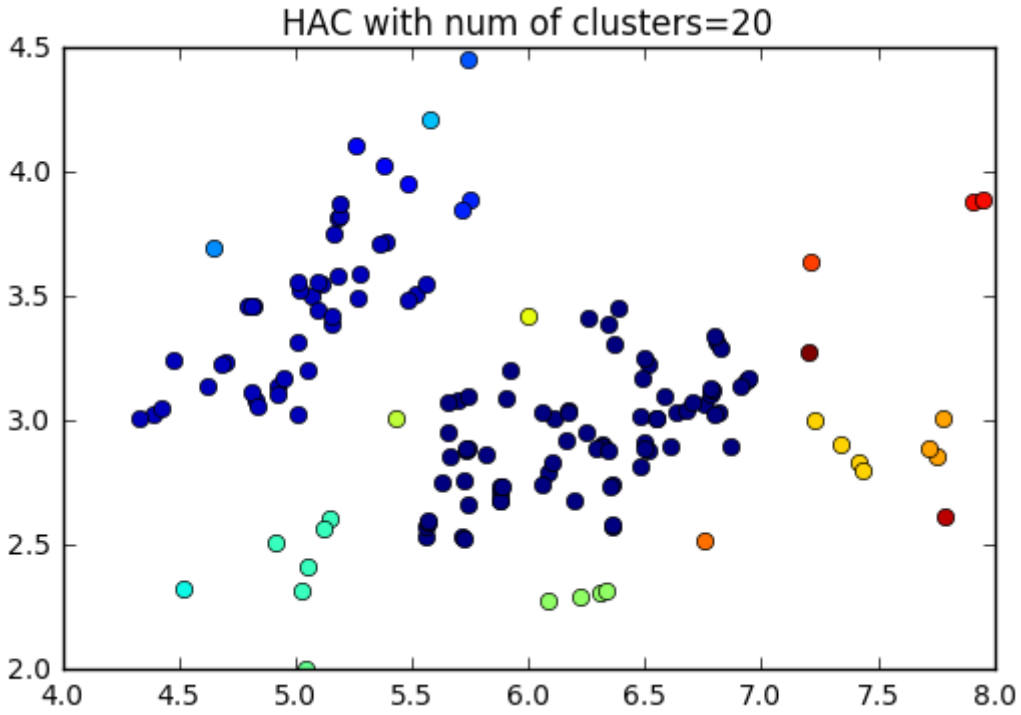
In [ ]:

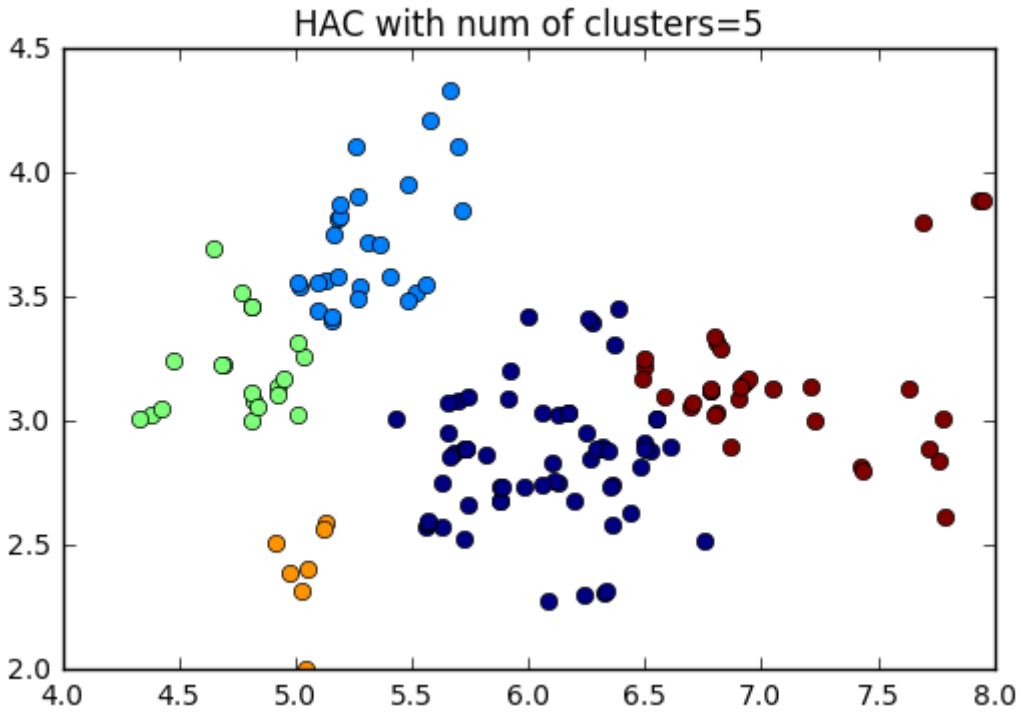
```
k++ initialization and for K=5 has the best score
```

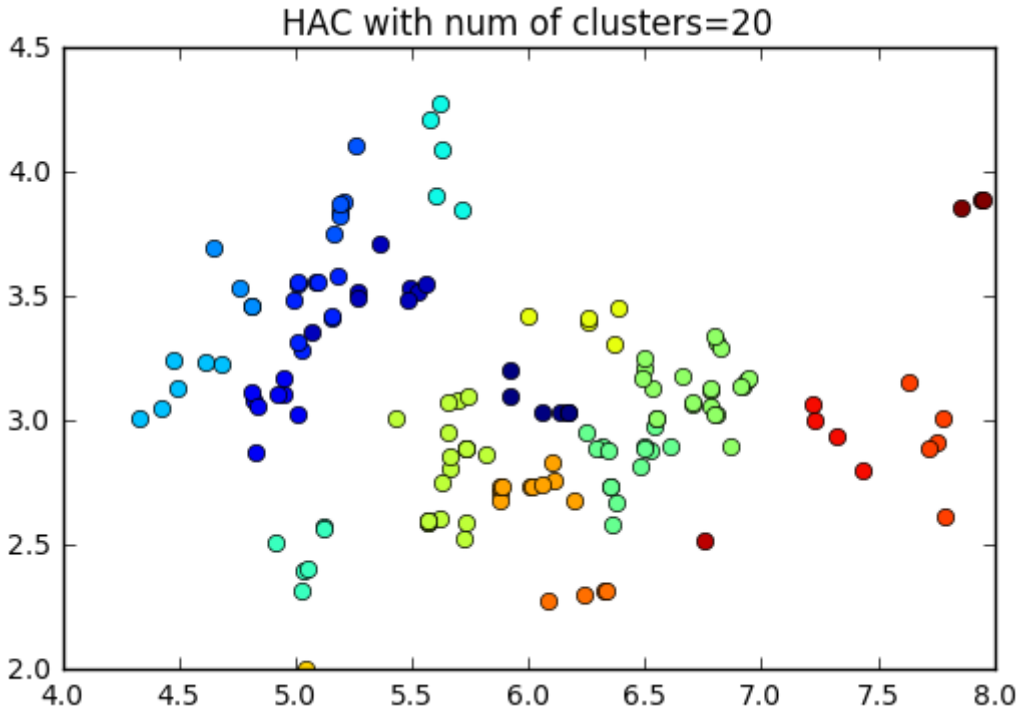
In [4]:

```
#1c
for i,j in enumerate(['min','max']):
    for m,k in enumerate([5,20]):
        agg,_=ml.cluster.agglomerative(X,k,j)
        plt.clf()
        ml.plotClassify2D(None,X,agg)
        #plt.xlabel('')
        #plt.ylabel('')
        if k==5:
            plt.title('HAC with num of clusters=5')
        else:
            plt.title('HAC with num of clusters=20')
        plt.show()
```


HAC with num of clusters=5
4.5
4.0
3.5
3.0
2.5
2.0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


HAC with num of clusters=20
4.5
4.0
3.5
3.0
2.5
2.0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



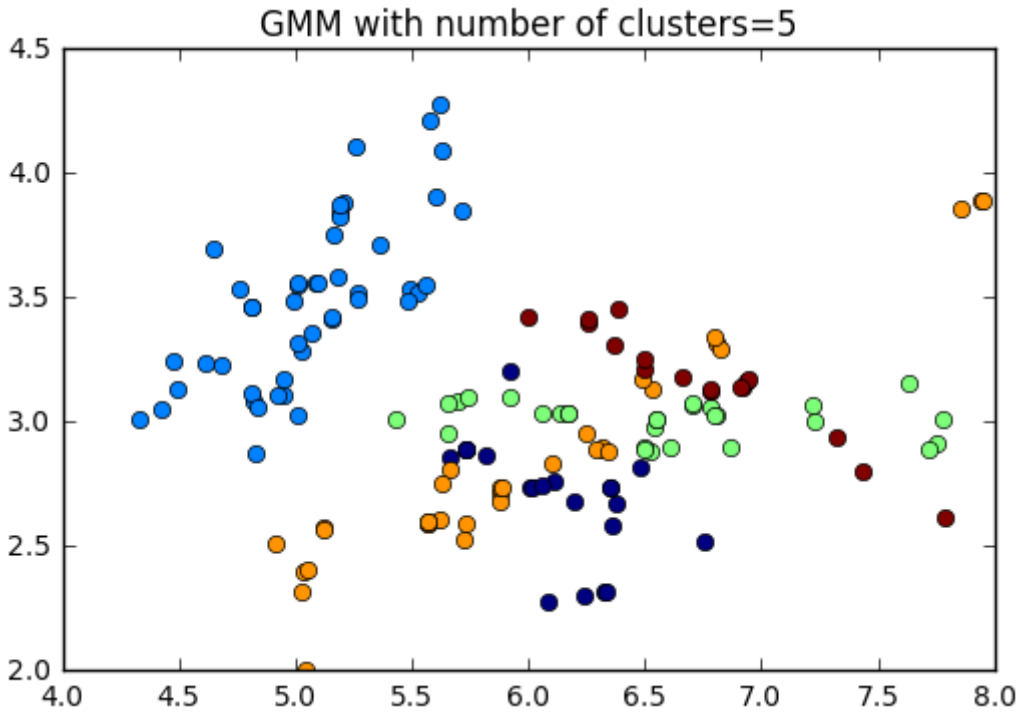


In [ ]:

From the graph it **is** evident that single linkage **is** good **for** stripe like (long continuo
us data)
and complete clustering **is** good **for** spherical clusters.
k means **is** good way of clustering **as** it has low computational complexity

In [7]:

```
#1d

z,m,n,_=ml.cluster.gmmEM(X,5)
plt.clf()
ml.plotClassify2D(None,X,z)
plt.title('GMM with number of clusters=5')
plt.show()
```



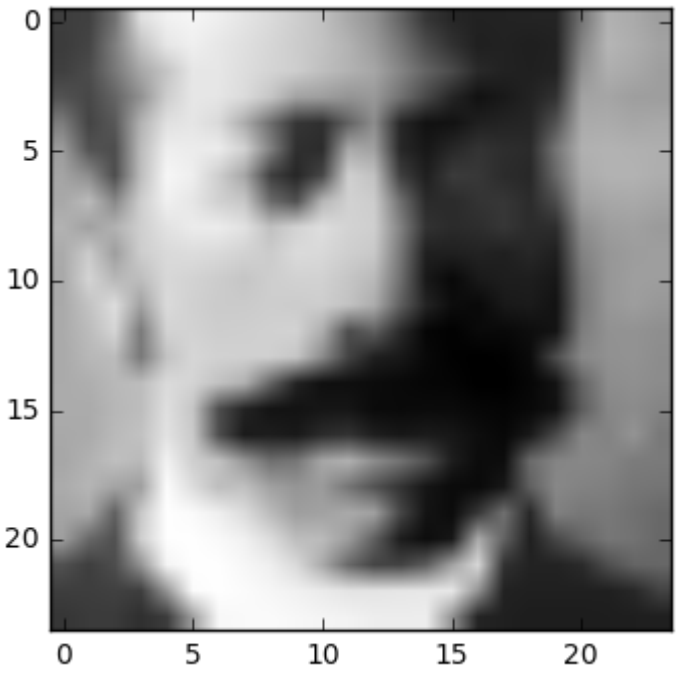
In [ ]:

```
The Agglomerative clustering h works well for simple and small dataset.
EM Gaussian mixture models are good for overlapping cluster dataset where a data might
not
be strictly associated with any cluster.Kmeans has low computational complexity.
```

In [57]:

```
#2a
X = np.genfromtxt("A5/data/faces.txt", delimiter=None) # load face dataset
mean = X.mean(axis=0)

X0= X-mean[:,]

print X0.shape

plt.figure()
#print i
plt.clf()
img = np.reshape(X0[1,:],(24,24)) # convert vectorized data point to 24x24 image patch
plt.imshow( img.T , cmap="gray")
plt.show()
```

```
(4916L, 576L)
```

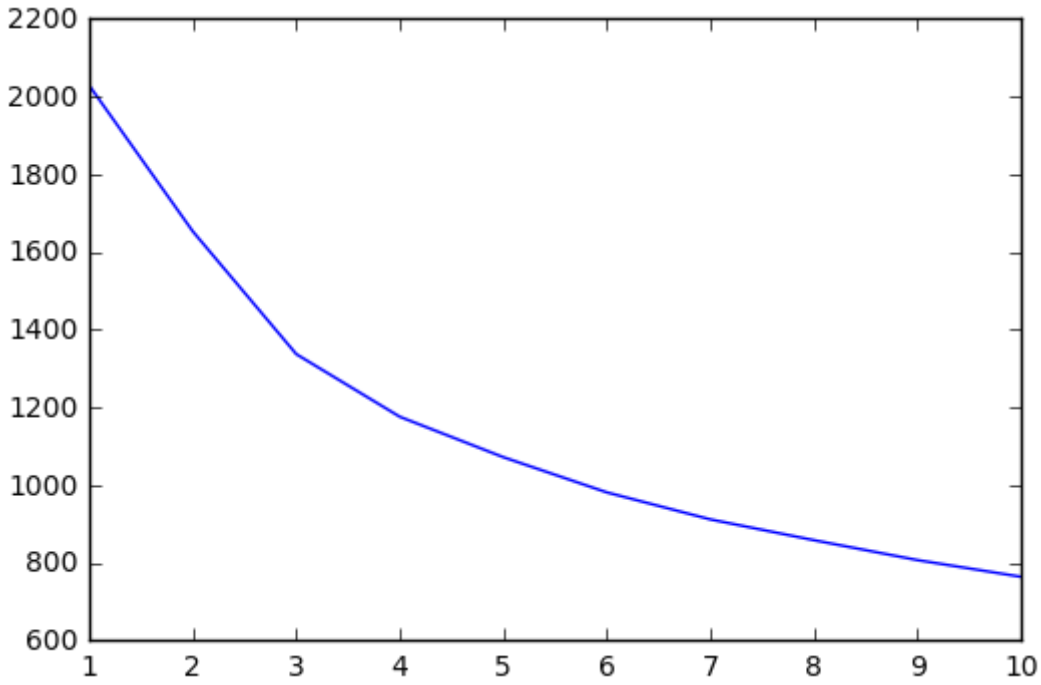


In [47]:

```
from scipy import linalg
u,s,vh= linalg.svd(X0,full_matrices=False)
w = u.dot(np.diag(s))
```

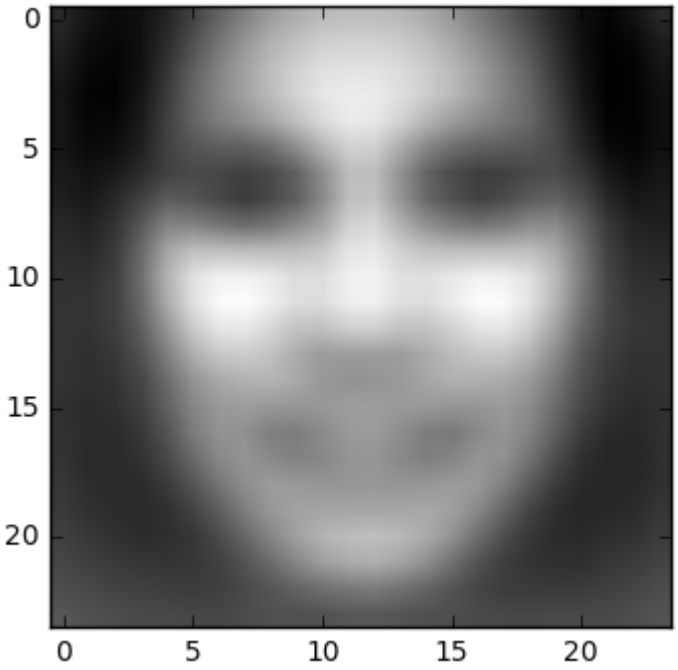
In [48]:

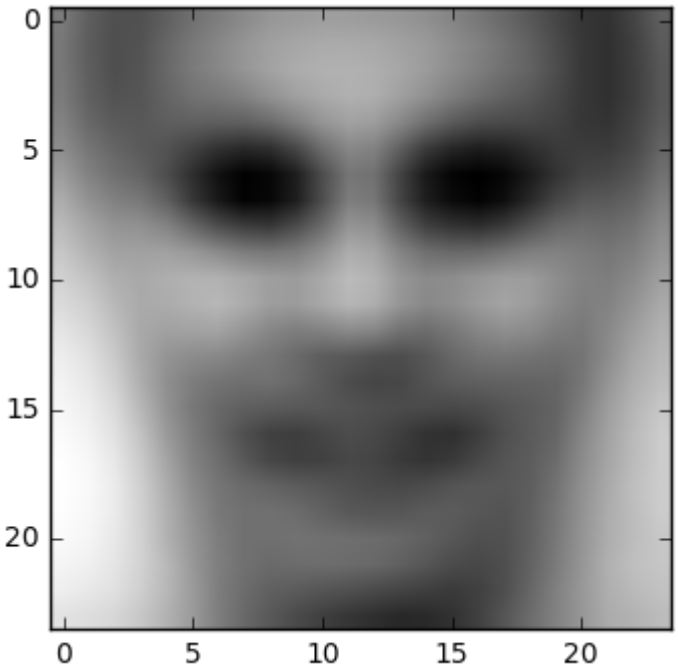
```
err=[None]*10

for i in range(1,11):
    x0 = w[:,:i].dot(vh[:i,:])
    err[i-1]=np.mean((X0-x0)**2)

plt.clf()
plt.plot(range(1,11),err)
#plt.ylim(0,2000)
plt.show()
```

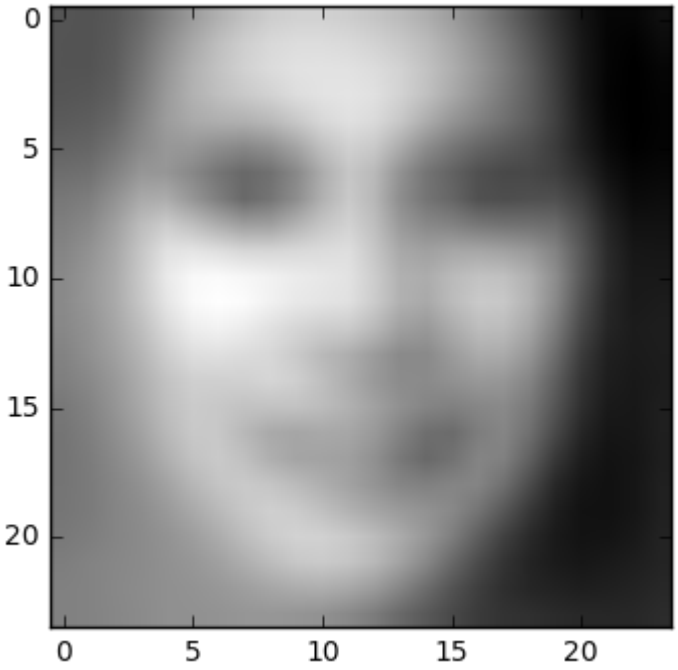


In [54]:

```
sf = 2*np.median(np.abs(w[:,0]))
img = np.reshape(mean+sf*vh[0,:],(24,24))
plt.clf()
plt.imshow( img.T , cmap="gray")
plt.show()

plt.clf()
sf = 2*np.median(np.abs(w[:,1]))
img = np.reshape(mean+sf*vh[1,:],(24,24))
plt.imshow( img.T , cmap="gray")
plt.show()

plt.clf()
sf = 2*np.median(np.abs(w[:,2]))
img = np.reshape(mean+sf*vh[2,:],(24,24))
plt.imshow( img.T , cmap="gray")
plt.show()
```

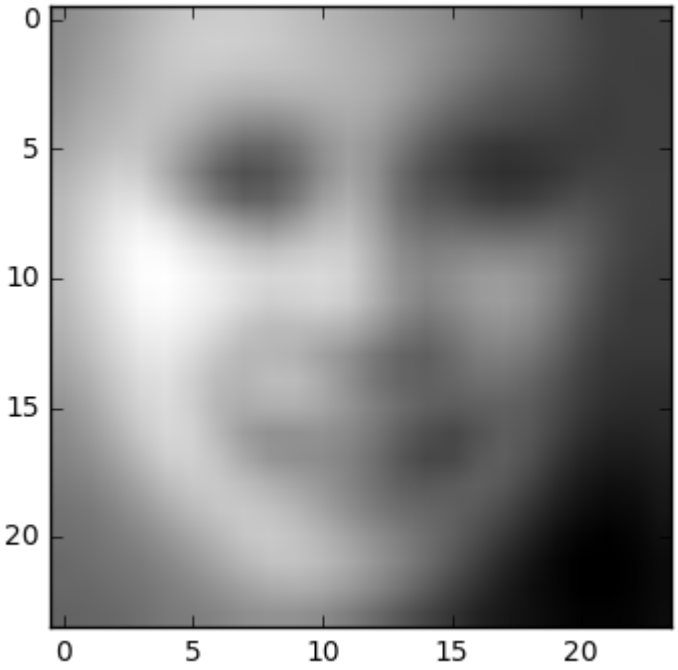

0
5
10
15
20
0
5
10
15
20

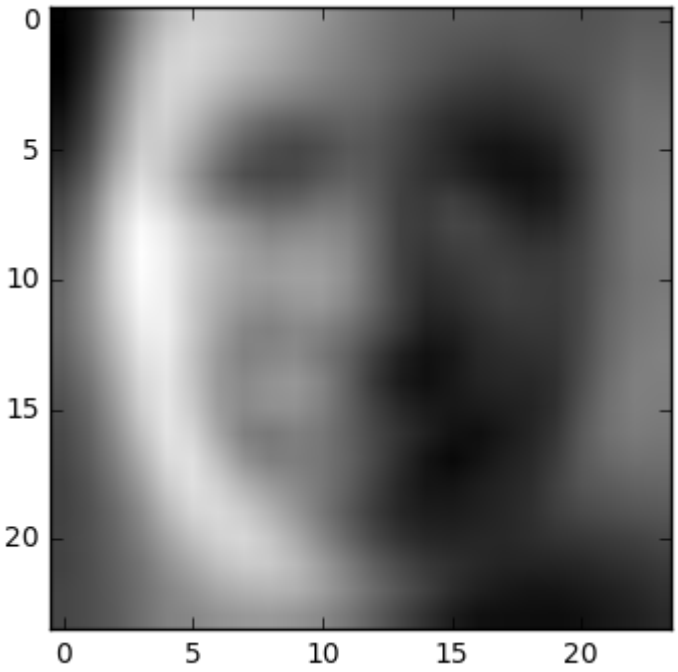

0
5
10
15
20
0
5
10
15
20

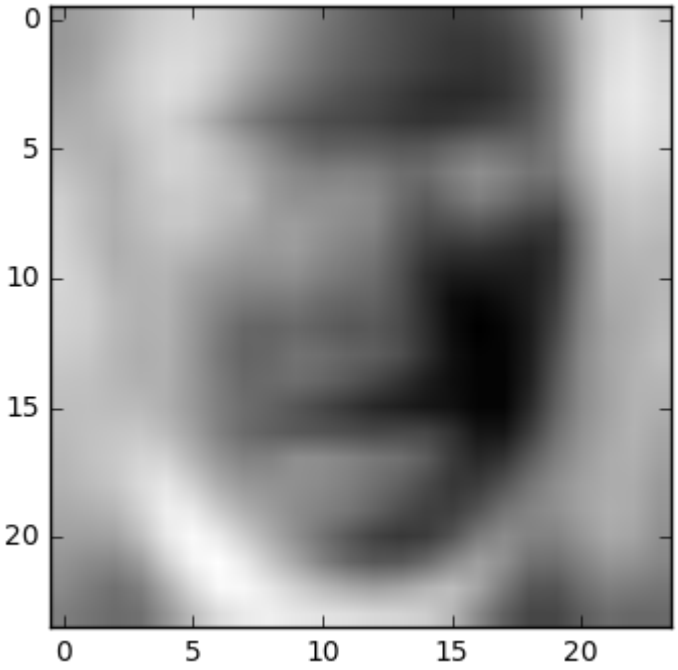

0
5
10
15
20
0
5
10
15
20

In [71]:

```
#2e
k=[5,10,50]
for i,j in enumerate(k):
    plt.clf()
    prin=mean+sf*vh[:j,:]
    img=w[1,:j].dot(prin)
    img = np.reshape(img,(24,24))
    plt.imshow( img.T , cmap="gray")
    plt.show()
```

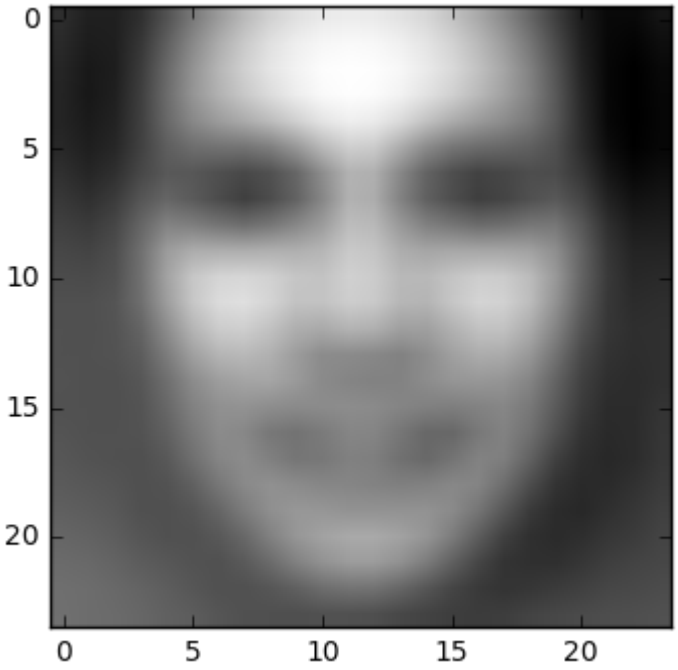

0
5
10
15
20
0
5
10
15
20

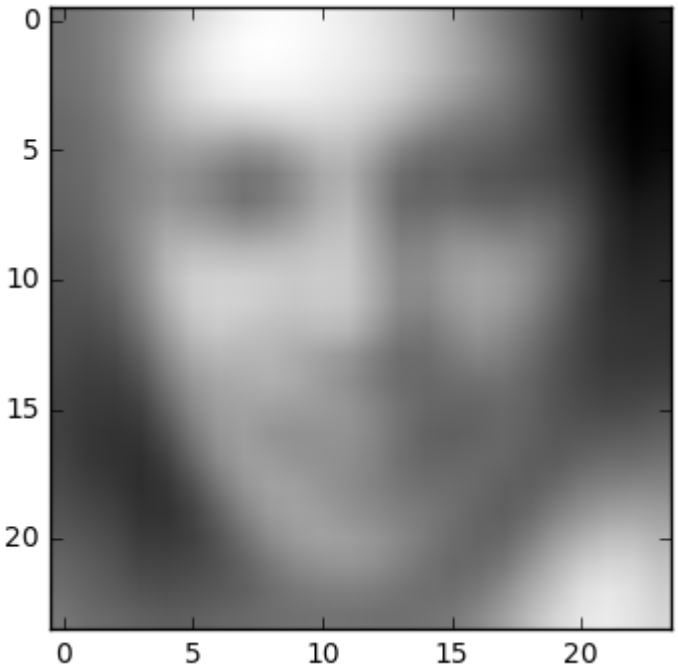

0
5
10
15
20
0
5
10
15
20

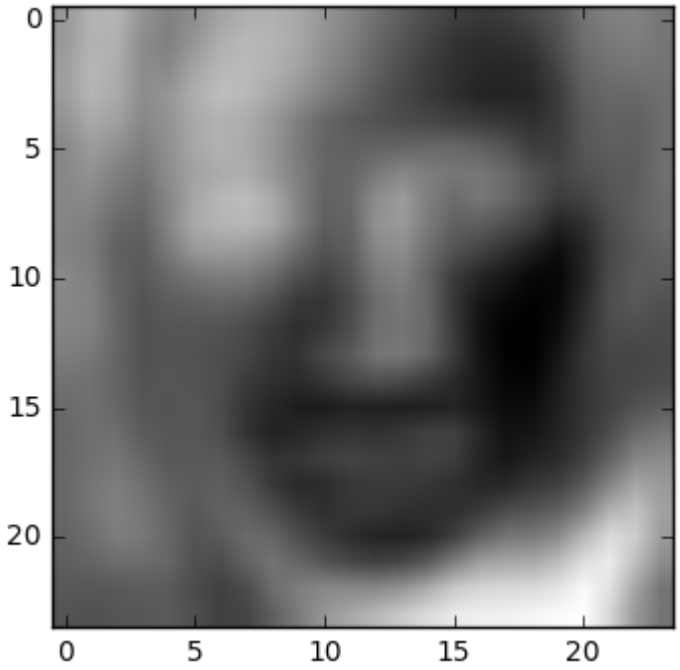

0
5
10
15
20
0
5
10
15
20

In [70]:

```
for i,j in enumerate(k):
    plt.clf()
    prin=mean+sf*vh[:j,:]
    img=w[0,:j].dot(prin)
    img = np.reshape(img,(24,24))
    plt.imshow( img.T , cmap="gray")
    plt.show()
```

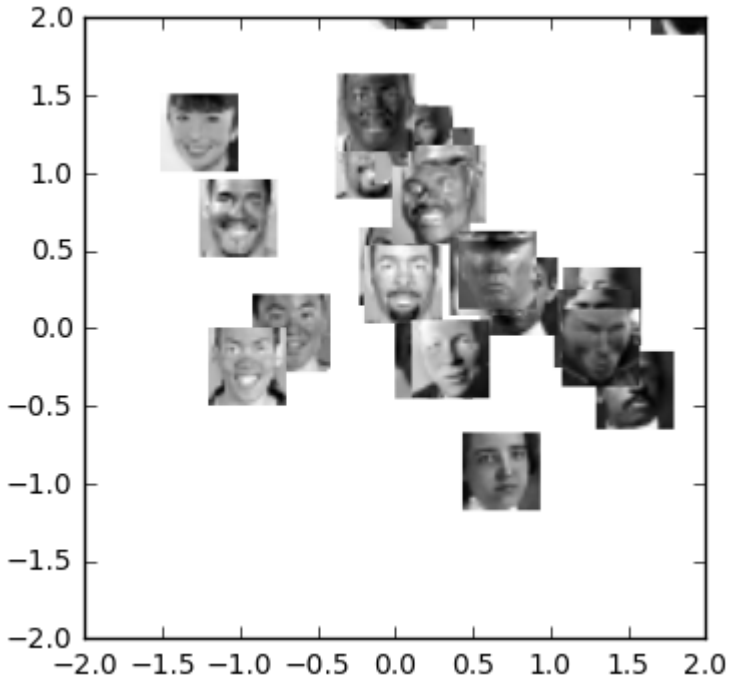

0
5
10
15
20
0
5
10
15
20


0
5
10
15
20
0
5
10
15
20



In [99]:

```
w = u.dot(np.diag(s))
idx = range(0,25)
import mltools.transforms
coord,params = ml.transforms.rescale( w[:,0:2] ) # normalize scale of "W" locations
for i in idx:
    loc = (coord[i,0],coord[i,0]+0.5, coord[i,1],coord[i,1]+0.5)
    img = np.reshape( X0[i,:], (24,24) )
    plt.imshow( img.T , cmap="gray", extent=loc ) # draw each image
    plt.axis( (-2,2,-2,2) )
plt.show()
```



In [ ]: