In [350]:
```
import mltools as ml
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
```

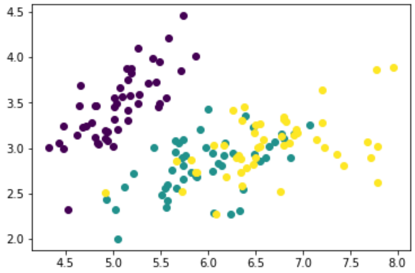
Problem 1

Part A

In [351]:
```
iris = np.genfromtxt("data/iris.txt",delimiter=None)
X = iris[:,0:2] #to get only the first two Features
Y = iris[:,-1]
print Y
print X
print X.shape
```

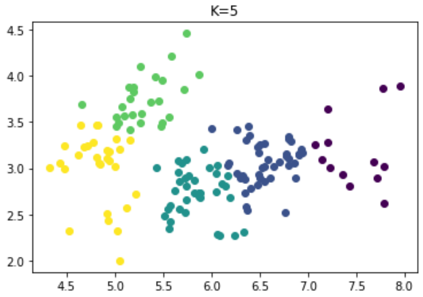
```
 [ 5.6655144  2.8548709]
 [ 7.7150636  2.8910809]
 [ 6.3479656  2.7361103]
 [ 6.795869   3.3222764]
 [ 7.2004658  3.2761173]
 [ 6.2933944  2.8924806]
 [ 6.1699019  3.033324 ]
 [ 6.4982575  2.89205  ]
 [ 7.2263808  3.002125 ]
 [ 7.4353972  2.8024043]
 [ 7.9528335  3.8910375]
 [ 6.4795806  2.8164015]
 [ 6.3434387  2.8831155]
 [ 6.1938172  2.6800039]
 [ 7.7829518  3.0143588]
 [ 6.3857928  3.4555813]
 [ 6.4916516  3.174443 ]
 [ 6.0621873  3.0340987]
 [ 6.9462665  3.1728895]
 [ 6.7812699  3.1332403]
```

In [352]:
```
ml.plotClassify2D(None, X,Y)
plt.show()
```



Part B

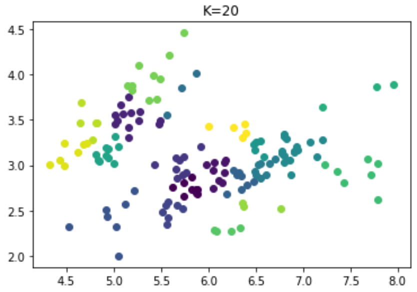
In [353]:
```
#Kneans returns z, c, sumd

d = np.inf
for i in range(10):
    z, c, sumd = ml.cluster.kmeans(X,K=5, init='random')
    if c < d:
        Z = z
        B2 = c
        d = sumd
plt.title("K=5")
ml.plotClassify2D(None, X,Z)
plt.show()
```



K : Number of clusters.

In [354]:

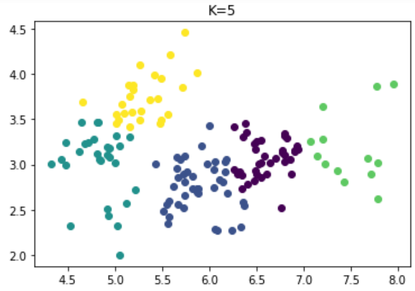
```
#Kneans returns z, c, sumd

d = np.inf
for i in range(10):
    z, c, sumd = ml.cluster.kmeans(X,K=20, init='random')
    if C < d:
        Z = z
        B2 = c
        d = sumd
plt.title("K=20")
ml.plotClassify2D(None, X,Z)
plt.show()
```



Next up, we are gonna run K means on the data 5+ times with different intitialization.

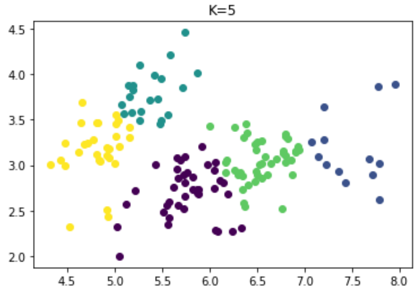
In [355]:

```
d = np.inf
for i in range(10):
    z, c, sumd = ml.cluster.kmeans(X,K=5, init='random')
    if C < d:
        Z = z
        B2 = c
        d = sumd
plt.title("K=5")
ml.plotClassify2D(None, X,Z)
plt.show()
```



In [356]:

```
d = np.inf
for i in range(10):
    z, c, sumd = ml.cluster.kmeans(X,K=5, init='farthest')
    if c < d:
        Z = z
        B2 = c
        d = sumd
plt.title("K=5")
ml.plotClassify2D(None, X,Z)
plt.show()
```

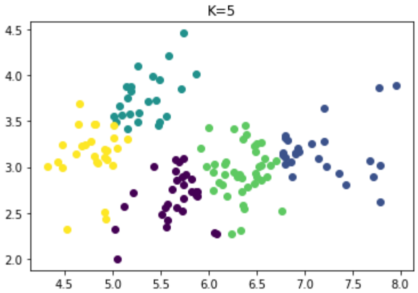


Here we choose the init = Farthest which basically chooses the first

cluster uniformly and then it chooses the point Farthest from the clusters.

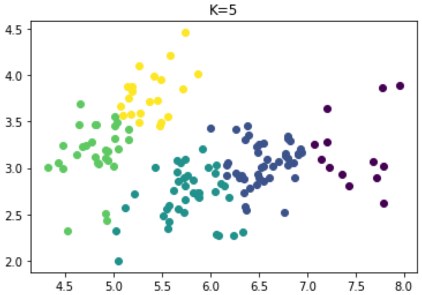
In [357]:

```
d = np.inf
for i in range(10):
    z, c, sumd = ml.cluster.kmeans(X,K=5, init='random')
    if c < d:
        Z = z
        B2 = c
        d = sumd
plt.title("K=5")
ml.plotClassify2D(None, X,Z)
plt.show()
```



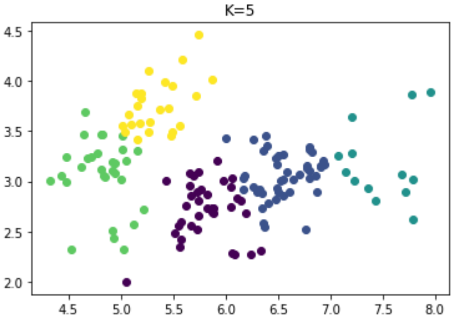
In [358]:

```
d = np.inf
for i in range(10):
    z, c, sumd = ml.cluster.kmeans(X,K=5, init='random')
    if C < d:
        Z = z
        B2 = c
        d = sumd
plt.title("K=5")
ml.plotClassify2D(None, X,Z)
plt.show()
```



In [359]:

```
d = np.inf
for i in range(10):
    z, c, sumd = ml.cluster.kmeans(X,K=5, init='k++')
    if C < d:
        Z = z
        B2 = c
        d = sumd
plt.title("K=5")
ml.plotClassify2D(None, X,Z)
plt.show()
```



Here we choose the init = k++ which basically chooses the first

cluster uniformly and then it chooses the point randomly proportional to distance from the current clusters

The issue is that kmeans has random ways to start the clustering procedure. In the graphs above I used the random, the Farthest and the k++ way to INITIATE the the clustering process. This causes the clutering to be different everytime.

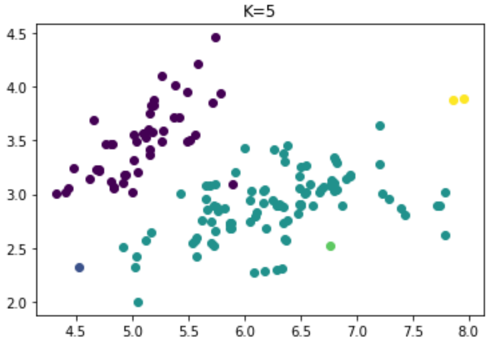
Part C

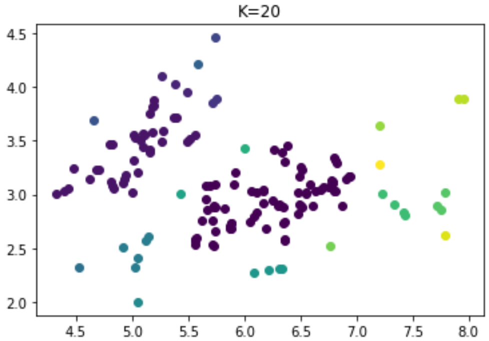
agglomerative returns z, join

Single Linkage

In [360]:

```
#k=5
z, join = ml.cluster.agglomerative(X, K=5, method='min')
#we use min for single linkage
plt.title("K=5")
ml.plotClassify2D(None,X,z)
plt.show()


#k=20
z, join = ml.cluster.agglomerative(X, K=20, method='min')
#we use min for single linkage
plt.title("K=20")
ml.plotClassify2D(None,X,z)
plt.show()
```



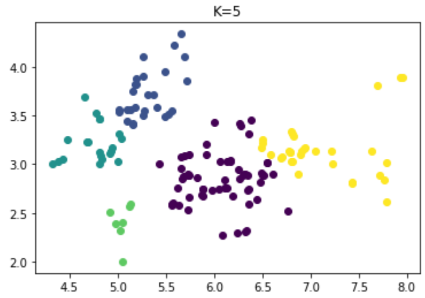


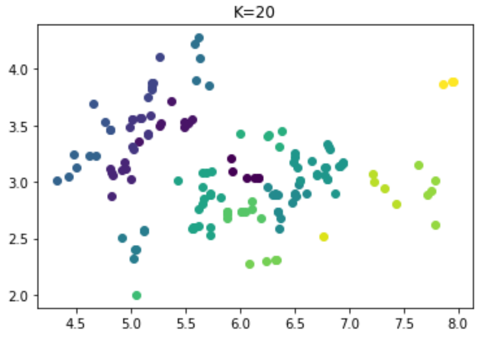
With single linkage the clustering has less similarity in terms of how much data points each cluster has. For instance, for the graph when k=5 we can see that there is a cluster made out of a single point.

Complete Linkage

In [361]:

```
#k=5
z, join = ml.cluster.agglomerative(X, K=5, method='max')
#we use max for single linkage
plt.title("K=5")
ml.plotClassify2D(None,X,z)
plt.show()


#k=20
z, join = ml.cluster.agglomerative(X, K=20, method='max')
#we use max for single linkage
plt.title("K=20")
ml.plotClassify2D(None,X,z)
plt.show()
```





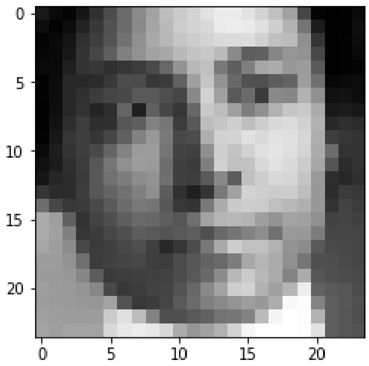
With complete linkage the data seems to be more equally distributed by the cluster.

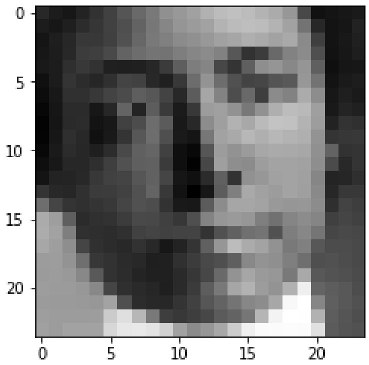
```
In [1]: import mltools as ml
        import numpy as np
        import matplotlib.pyplot as plt
        np.random.seed(0)
        import scipy
        from scipy import linalg
```

Problem 2

Part A

```
In [2]: X = np.genfromtxt("data/faces.txt",delimiter=None)
        mu = X.mean(axis=0, keepdims=True)
        #print mu
        X_0 = X-mu


        img = np.reshape(X[3,:],(24,24))
        img1 = np.reshape(X_0[3,:],(24,24))
        plt.imshow(img.T, cmap="gray")
        plt.show()

        plt.imshow(img1.T, cmap="gray")
        plt.show()
```



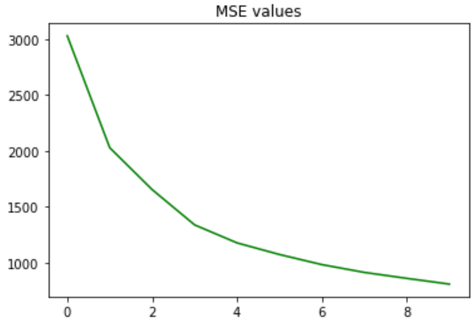


```
U,S,V = scipy.linalg.svd(X_0, False)
W = U.dot(np.diag(S))
print U.shape
print S.shape
print W.shape
print V.shape
print W
```

```
(4916L, 576L)
(576L,)
(4916L, 576L)
(576L, 576L)
[[  8.18179734e+02  -5.59694010e+01   3.05239619e+02 ...,   5.23234822e+00
   -5.24999420e+00  -2.45927306e+00]
 [ -2.23936743e+02   4.31082278e+02   9.19104386e+02 ...,  -4.30026085e+00
   -1.19454998e+00   1.25611925e+00]
 [  3.26559700e+02  -5.48923241e+02   6.36293364e+01 ...,  -4.91201271e-01
   -9.37783859e-01  -9.30071561e-01]
 ...,
 [  1.51629405e+03  -3.53137572e+01   6.49234746e+00 ...,   1.59237349e+00
    3.85732508e-01  -1.07747364e+00]
 [ -2.42782896e+02  -6.05339905e+02  -1.37039083e+02 ...,   7.77060365e-01
    1.64704085e-01   1.16868877e+00]
 [ -1.76946498e+03   3.75137876e+02  -4.43782075e+02 ...,  -1.56431420e+00
    7.67654999e-03   2.48635833e-01]]
```

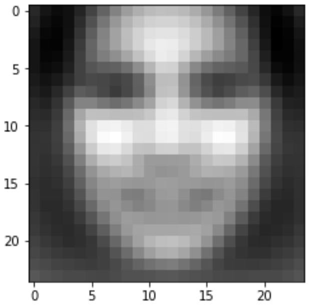
Part C

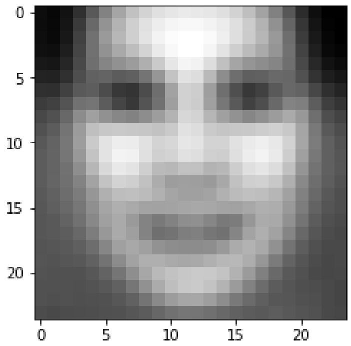
```
err = [None]*10
for k in range(10):
    Xhat0 = W[:,:k].dot( V[:k,:] )
    err[k] = np.mean((X_0-Xhat0)**2)
plt.title("MSE values")
plt.plot(err,'g-')
plt.show()
```

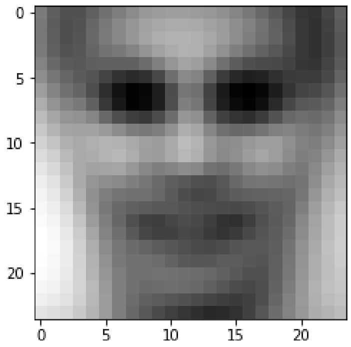


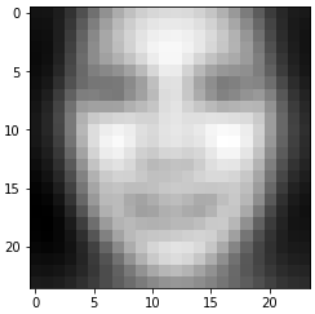
Part D

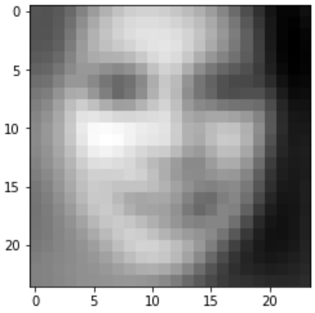
```
for j in range(3):
    a = 2*np.median( np.abs( W[:,j] ))
    #a = alpha and alpha is the scalar factor
    image_1 = np.reshape(mu + a*V[j,:], (24,24))
    image_2 = np.reshape(mu - a*V[j,:], (24,24))

    plt.imshow(image_1.T, cmap="gray")
    plt.show()
    plt.imshow(image_2.T, cmap="gray")
    plt.show()
```

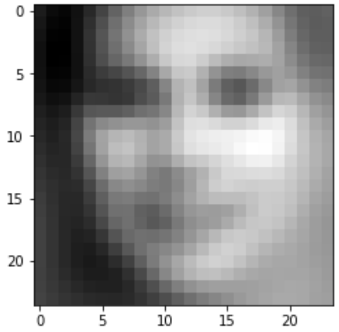












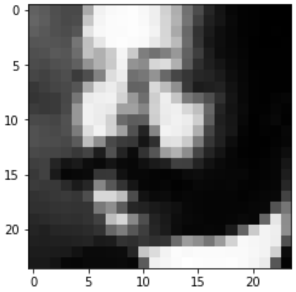
```
import mltools as ml
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
import scipy
from scipy import linalg
```

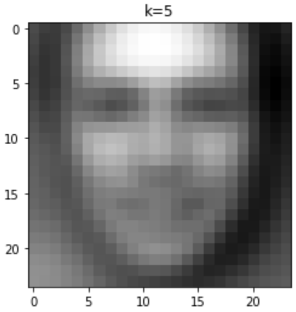
Part E

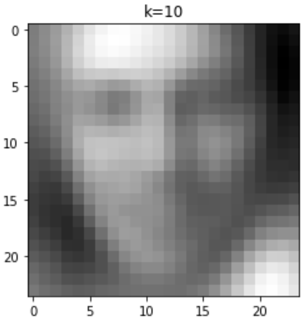
```
X = np.genfromtxt("data/faces.txt",delimiter=None)
mu = X.mean(axis=0, keepdims=True)
X_0 = X-mu
U,S,V = scipy.linalg.svd(X_0, False)
W = U.dot(np.diag(S))
```

1st face

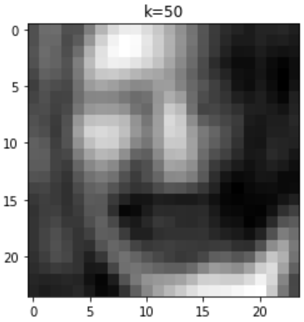
```
#
image = np.reshape(X[0,:], (24,24))
plt.imshow(image.T, cmap="gray")
plt.show()

#for k=5
image1 = mu + W[0,0:5].dot(V[0:5,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=5")
plt.imshow(image1.T, cmap="gray")
plt.show()

#for k=10
image1 = mu + W[0,0:10].dot(V[0:10,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=10")
plt.imshow(image1.T, cmap="gray")
plt.show()

#for k=50
image1 = mu + W[0,0:50].dot(V[0:50,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=50")
plt.imshow(image1.T, cmap="gray")
plt.show()

#for k=100
image1 = mu + W[0,0:100].dot(V[0:100,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=100")
plt.imshow(image1.T, cmap="gray")
plt.show()
```



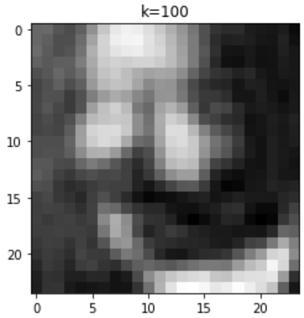




k=10



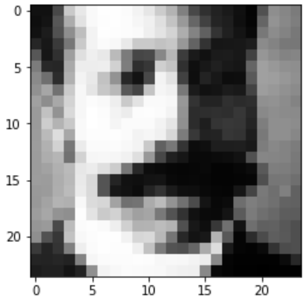
k=50

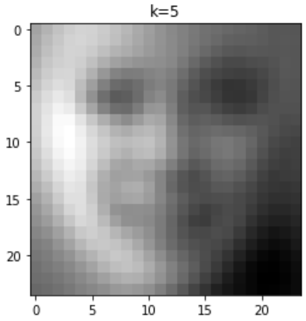


k=100

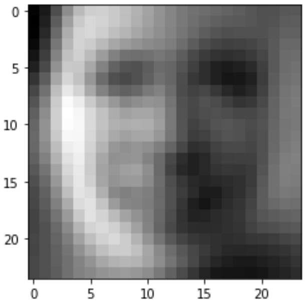
2nd Face

```
#
image = np.reshape(X[1,:], (24,24))
plt.imshow(image.T, cmap="gray")
plt.show()

#for k=5
image1 = mu + W[1,0:5].dot(V[0:5,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=5")
plt.imshow(image1.T, cmap="gray")
plt.show()

#for k=10
image1 = mu + W[1,0:10].dot(V[0:10,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=10")
plt.imshow(image1.T, cmap="gray")
plt.show()

#for k=50
image1 = mu + W[1,0:50].dot(V[0:50,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=50")
plt.imshow(image1.T, cmap="gray")
plt.show()

#for k=100
image1 = mu + W[1,0:100].dot(V[0:100,:])
image1 = np.reshape(image1,(24,24))
plt.title("k=100")
plt.imshow(image1.T, cmap="gray")
plt.show()
```

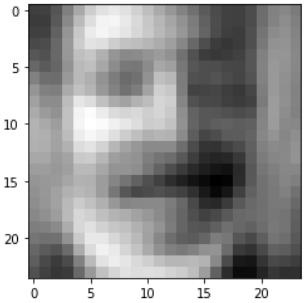




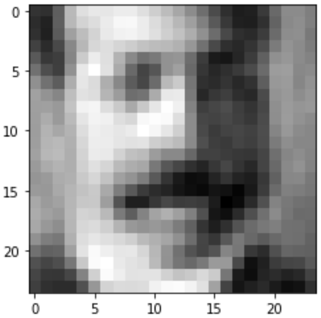
k=10



k=50



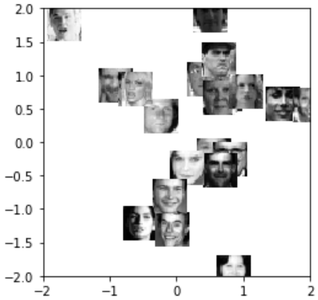
k=100



Part F

```
idx = np.floor(4000*np.random.rand(20))
idx = idx.astype('int')

# pick some data at random or otherwise; a list / vector of integer indices
import mltools.transforms
coord,params = ml.transforms.rescale( W[:,0:2] ) # normalize scale of "W" locations
plt.figure();
plt.show()
#plt.hold(True); # you may need this for pyplot
for i in idx:
# compute where to place image (scaled W values) & size
    loc = (coord[i,0],coord[i,0]+0.5, coord[i,1],coord[i,1]+0.5)
    img = np.reshape( X[i,:], (24,24) ) # reshape to square
    plt.imshow( img.T , cmap="gray", extent=loc ) # draw each image
    plt.axis( (-2,2,-2,2) )
plt.show()
```

```
<matplotlib.figure.Figure at 0xdcd1da0>
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2