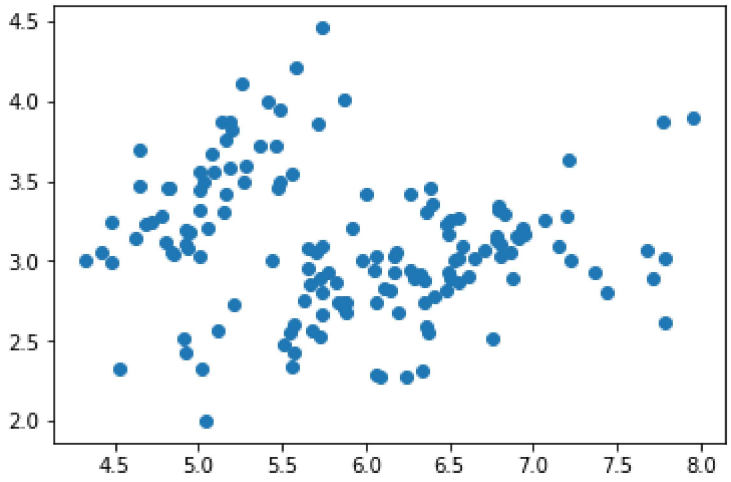
1 Clustering

In [3]:
```
from __future__ import division # For python 2.*

import numpy as np
import matplotlib.pyplot as plt
import mltools as ml

from scipy.linalg import svd

np.random.seed(0)
%matplotlib inline
```

Part 1

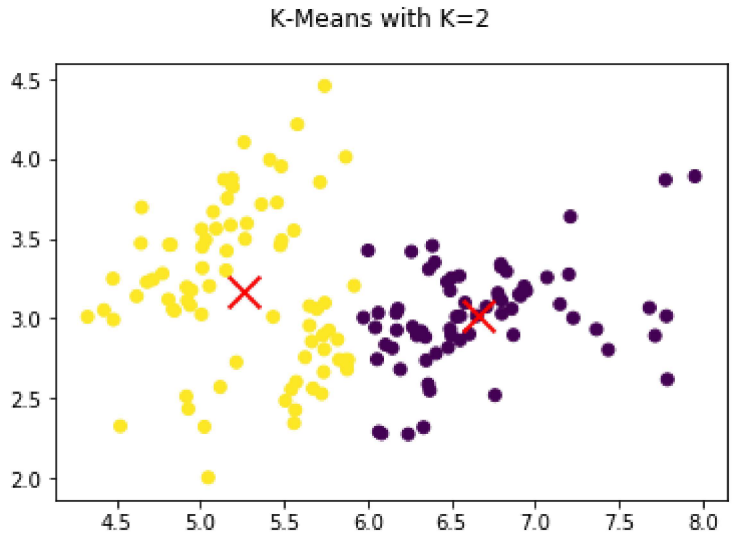
In [4]:
```
iris = np.genfromtxt("data/iris.txt",delimiter=None)
X = iris[:,0:2]

plt.scatter(X[:,0], X[:,1])
plt.show()
```



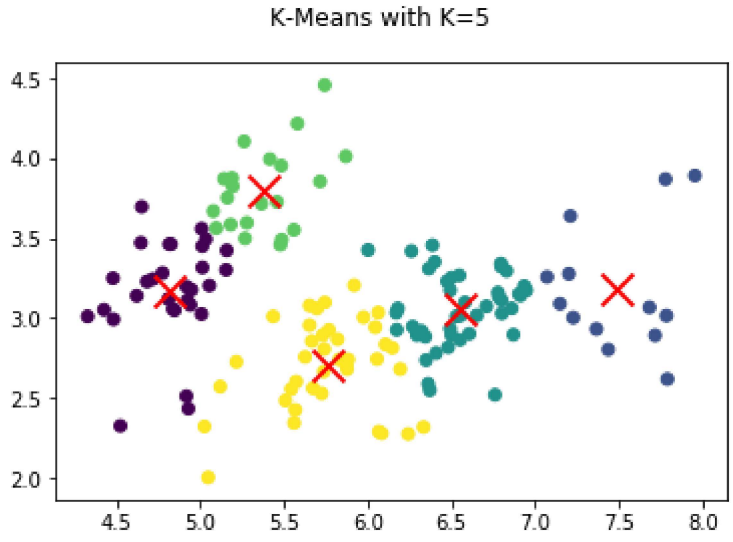
It appears to me as if there are about 5 distinct clusters.

Part 2

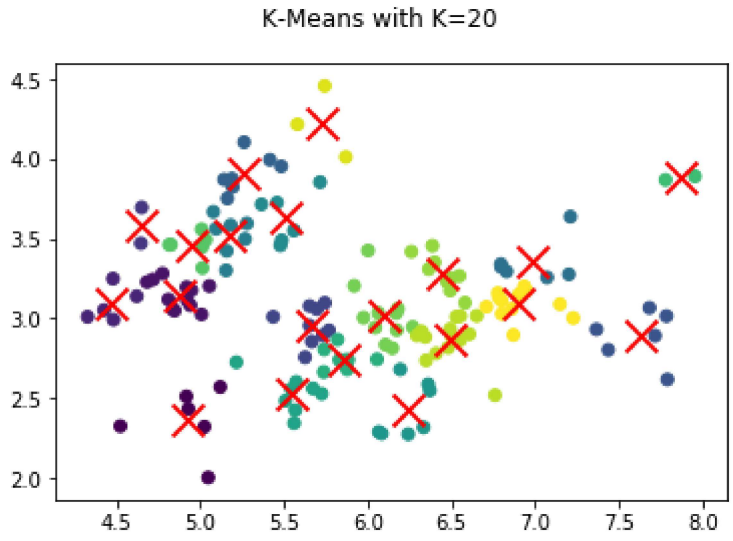
In [5]:
```
k = [2,5,20]
init = ['random','random','random','farthest','k++',]

Z = {}
mu = {}
ssd = {}
ini = {}

for i in k:
    seed = 1
    z_best = None
    m_best = None
    s_best = 999999999999999999
    i_best = None
    for j in init:
        np.random.seed(seed)
        seed += 20
        z, m, s = ml.cluster.kmeans(X, K=i, init=j, max_iter=100)
        if s < s_best:
            z_best = z
            m_best = m
            s_best = s
            i_best = j
    Z[i] = z_best
    mu[i] = m_best
    ssd[i] = s_best
    ini[i] = i_best

fig, ax = plt.subplots(1)
ax.scatter(X[:, 0], X[:, 1], c=Z[2]) # Plotting the data
ax.scatter(mu[2][:, 0], mu[2][:, 1], s=250, marker='x', facecolor='red', lw=2) # 
plt.suptitle('K-Means with K=2')
plt.show()
print "init type =",ini[2]
print "ssd =", ssd[2]

fig, ax = plt.subplots(1)
ax.scatter(X[:, 0], X[:, 1], c=Z[5]) # Plotting the data
ax.scatter(mu[5][:, 0], mu[5][:, 1], s=250, marker='x', facecolor='red', lw=2) # 
plt.suptitle('K-Means with K=5')
plt.show()
print "init type =",ini[5]
print "ssd =", ssd[5]

fig, ax = plt.subplots(1)
ax.scatter(X[:, 0], X[:, 1], c=Z[20]) # Plotting the data
ax.scatter(mu[20][:, 0], mu[20][:, 1], s=250, marker='x', facecolor='red', lw=2) 
plt.suptitle('K-Means with K=20')
plt.show()
print "init type =",ini[20]
print "ssd =", ssd[20]

np.random.seed(0)
```




```
init type = random
ssd = 57.877648397
```




```
init type = farthest
ssd = 21.0902063019
```

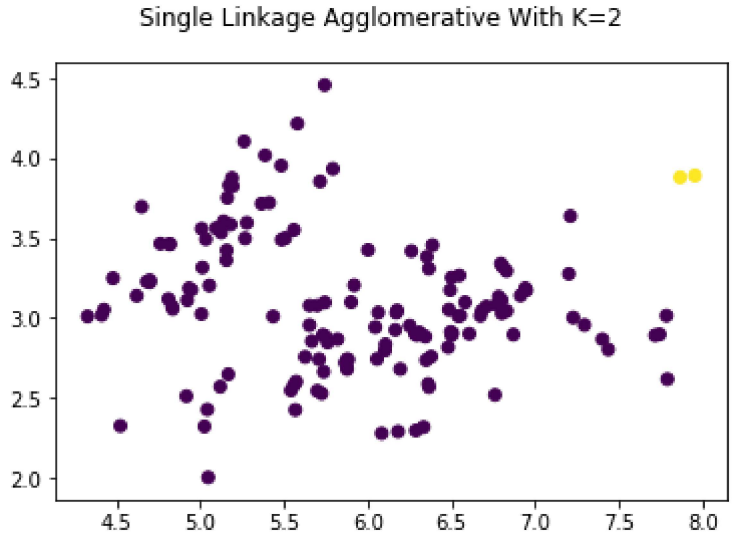


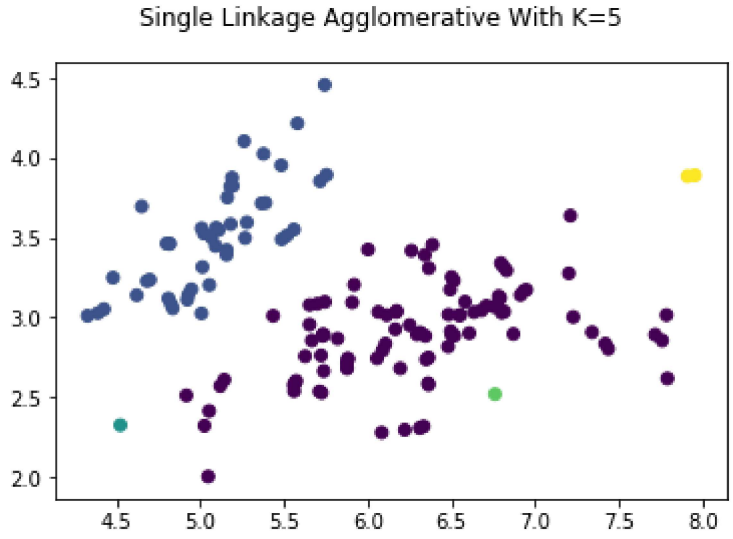

```
init type = random
```

```
ssd = 4.31990342373
```

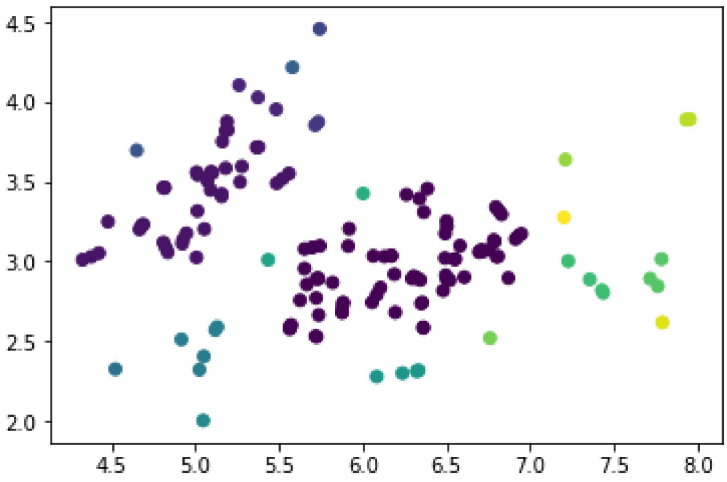
Part 3

In [6]:
```
k = [2,5,20]

for i in k:
    zi, q = ml.cluster.agglomerative(X, K=i, method='min')

    fig, ax = plt.subplots(1)
    ax.scatter(X[:, 0], X[:, 1], c=zi) # Plotting the data
    plt.suptitle('Single Linkage Agglomerative With K='+str(i))
    plt.show()

for i in k:
    zi, q = ml.cluster.agglomerative(X, K=i, method='max')

    fig, ax = plt.subplots(1)
    ax.scatter(X[:, 0], X[:, 1], c=zi) # Plotting the data
    plt.suptitle('Complete Linkage Agglomerative With K='+str(i))
    plt.show()
```

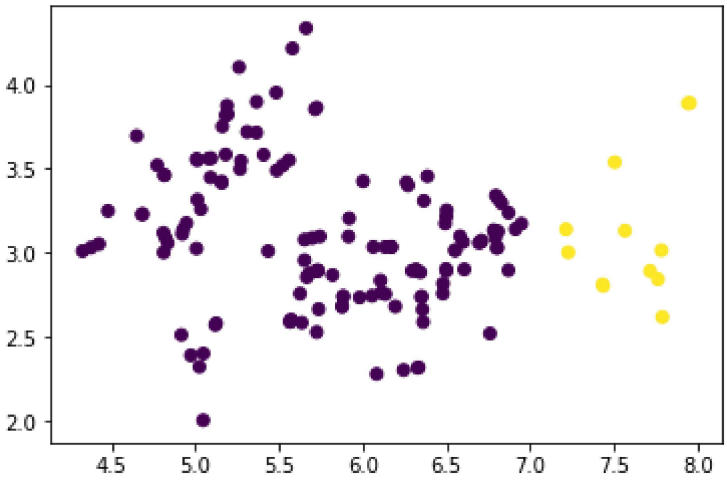




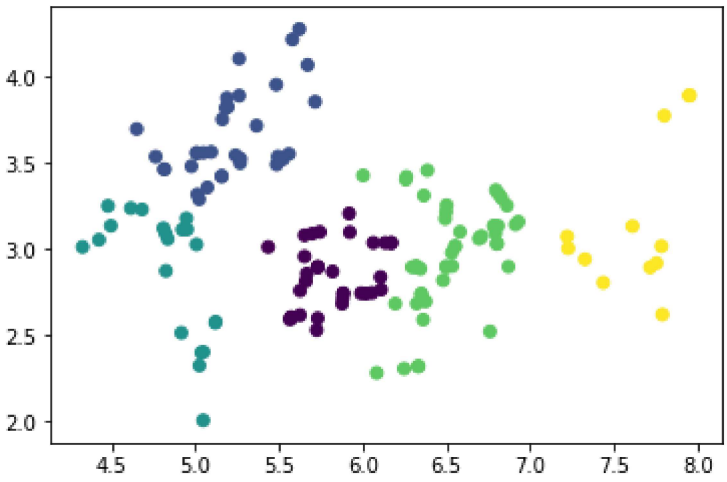
Single Linkage Agglomerative With K=20

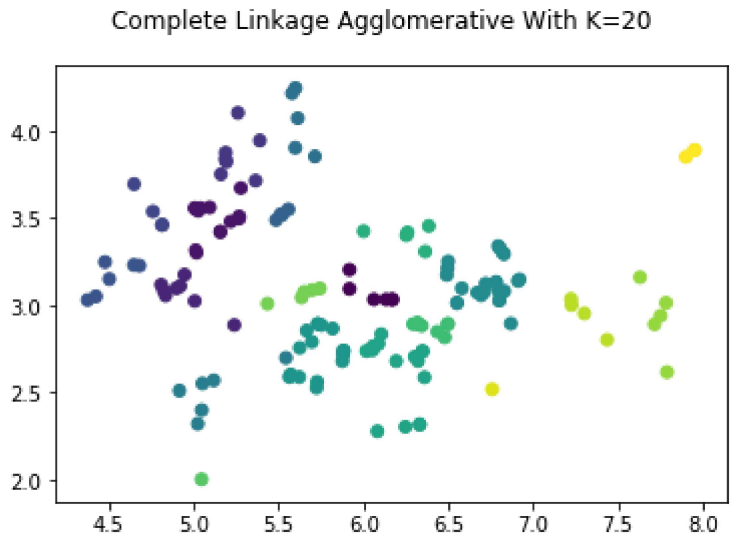


Complete Linkage Agglomerative With K=2



Complete Linkage Agglomerative With K=5



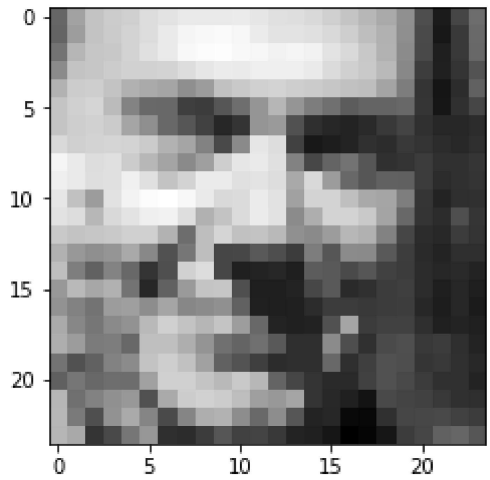


Part 4

Comparing the two methods, it seems that overall, k-means generally does better than agglomerative. K-means seems to, regardless of what the k is, give something that makes sense, and is a reasonable output for that k. Agglomerative clustering on the other hand, especially with Single Linkage is just atrocious. The only time it appears to do well is if the chosen value of k is close to an ideal number of clusters, and in that case, with Complete Linkage, it actually subjectively did just as well as k-means.

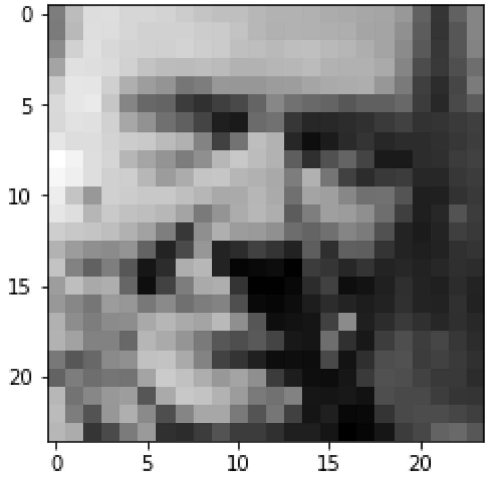
2 EigenFaces

In [7]:
```
X = np.genfromtxt("data/faces.txt", delimiter=None)
plt.figure()
img = np.reshape(X[911,:],(24,24))
plt.imshow( img.T , cmap="gray")
plt.show()
```



Part 1

In [8]:
```
mu = np.mean(X, axis=0)
X0 = X - mu

plt.figure()
img = np.reshape(X0[911,:],(24,24))
plt.imshow( img.T , cmap="gray")
plt.show()
```



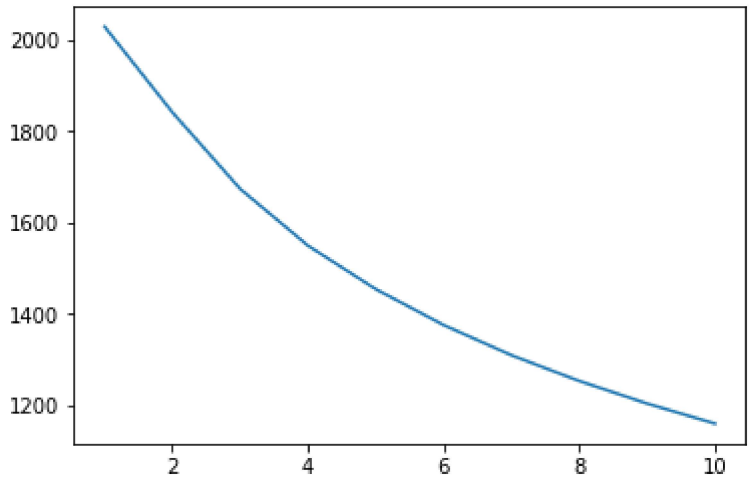
Part 2

In [9]:
```
U, s, V = svd(X0, full_matrices=False)

W = np.dot(U, np.diag(s))

print 'Shape of W = (%d, %d)' % W.shape, 'Shape of V = (%d, %d)' % V.shape
```

```
Shape of W = (4916, 576) Shape of V = (576, 576)
```

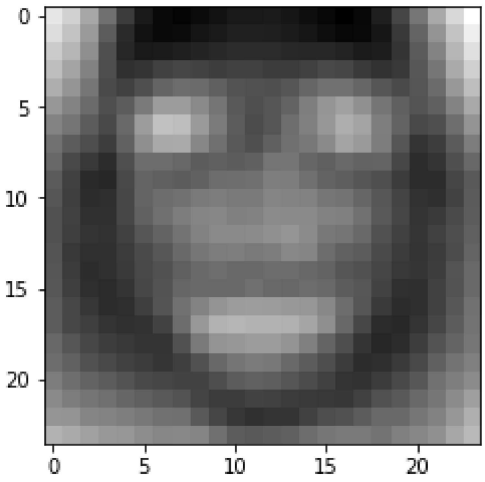
Part 3

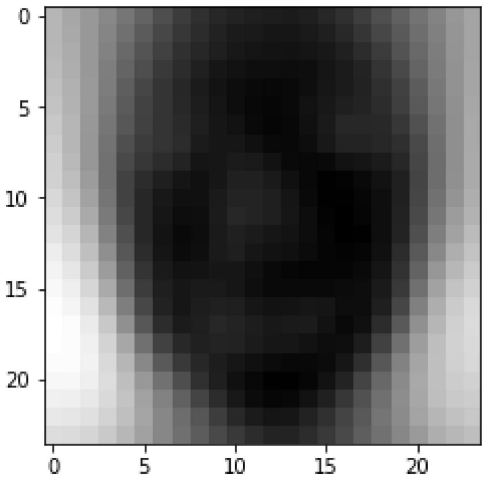
In [10]:
```
k = [i for i in range(1,11,1)]

Xhat = []
MSE = []

for i in k:
    Xhat.append(np.dot(W[:,:i], V[:i,:]))
    MSE.append(np.mean((X0-Xhat)**2))

plt.plot(k,MSE)
plt.show()
```

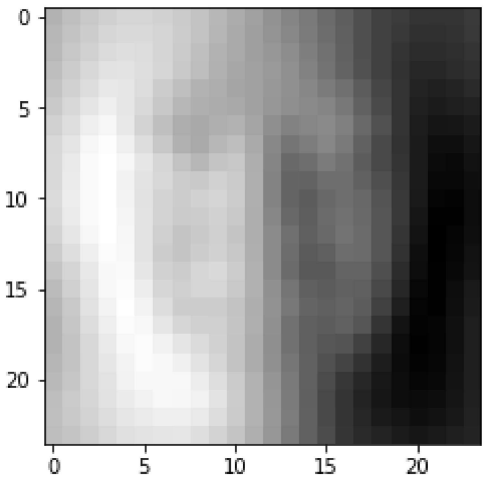


Part 4

In [34]:
```
for j in range(3):
    a = 2 * np.median(np.abs(W[:, j]))

    p1 = mu - a * V[j,:]
    p2 = mu + a * V[j,:]

    img = np.reshape(p2-p1,(24,24))
    plt.imshow( img.T , cmap="gray")
    plt.show()
```

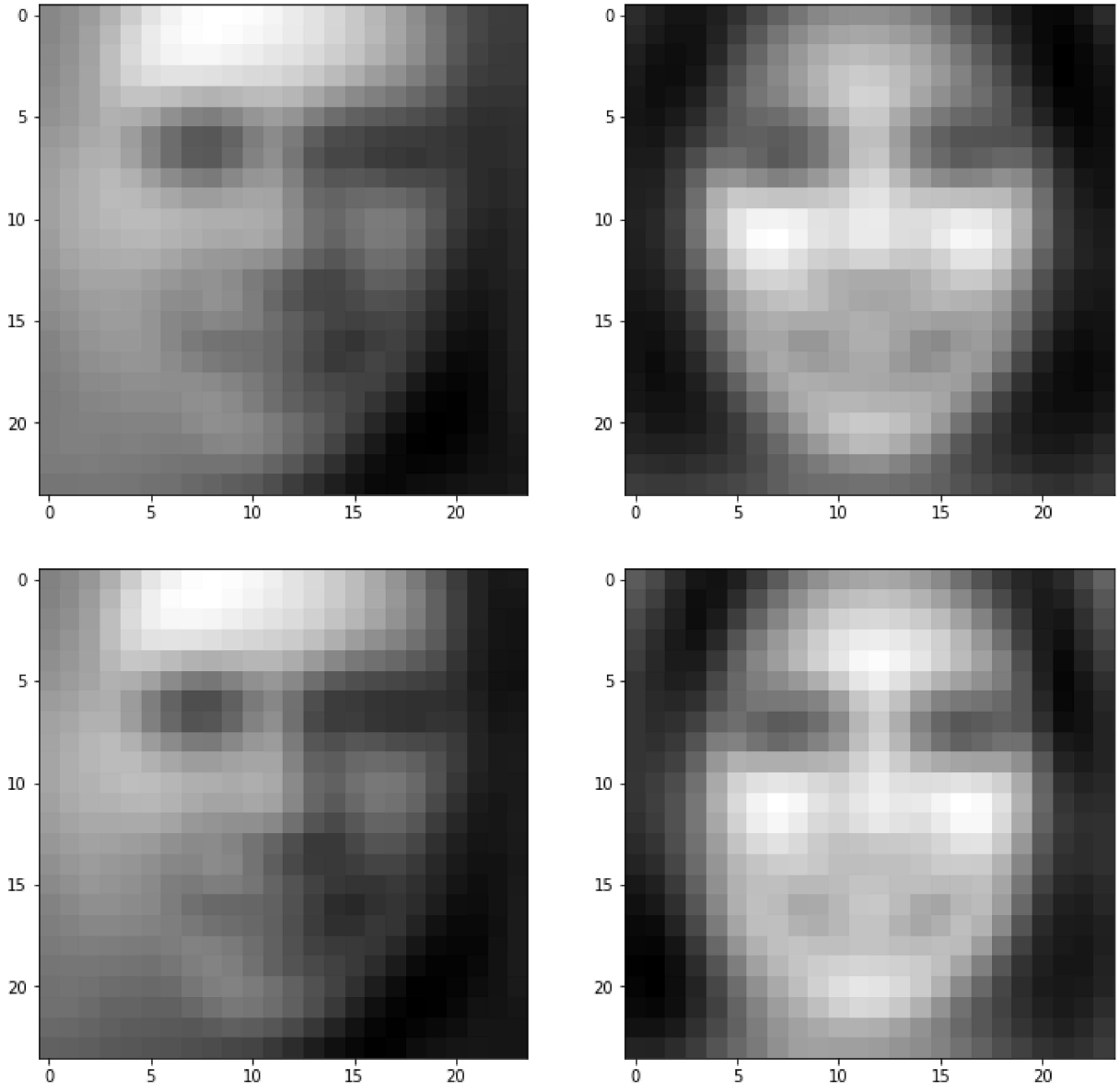


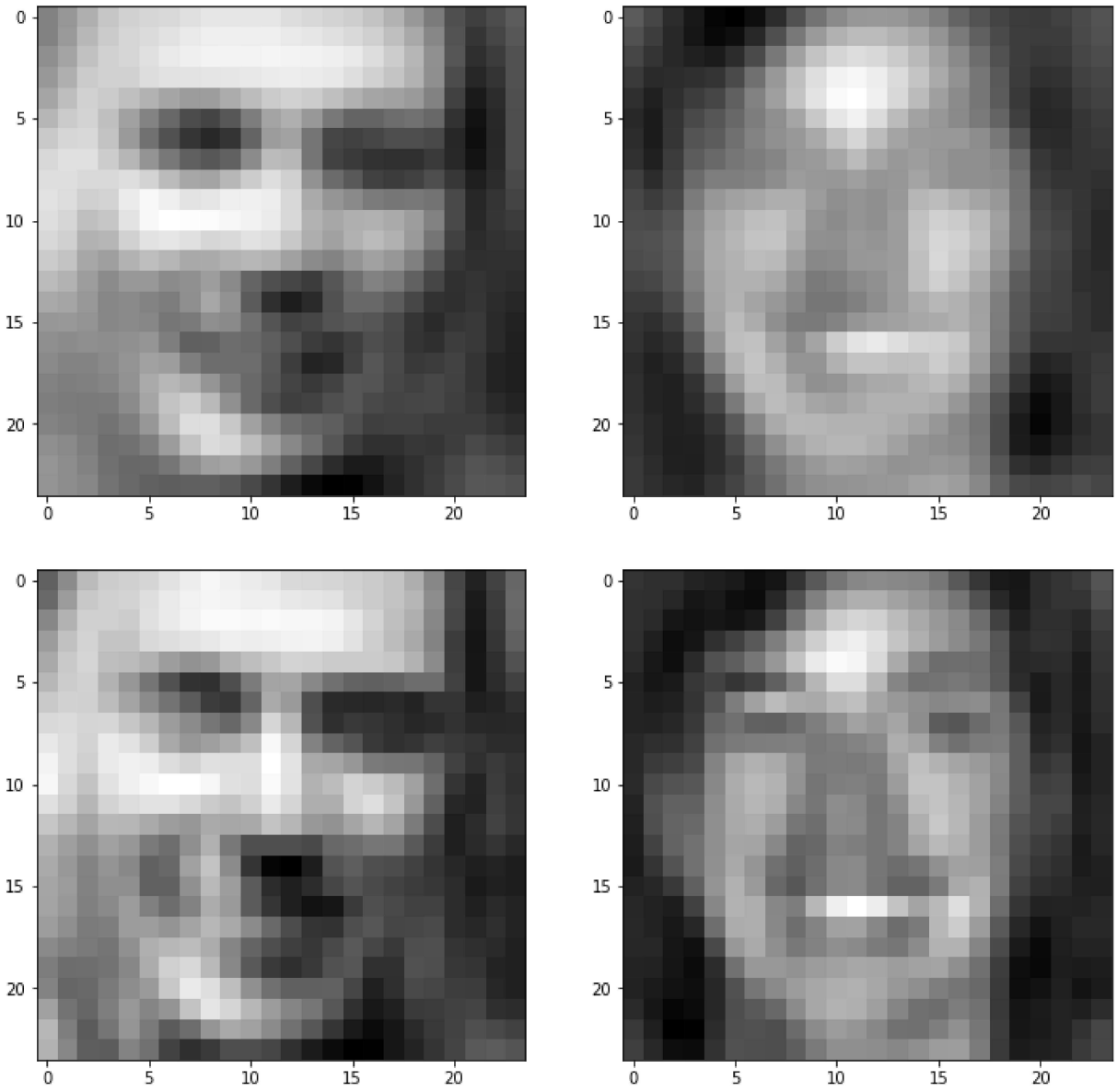




Part 5

In [41]:
```
k = [5,10,50,100]

for x in k:
    f, ax = plt.subplots(1, 2, figsize=(12, 15))

    idx = [911, 69]
    for j in range(len(idx)):
        i = idx[j]

        img = np.dot(W[i, :x], V[:x]) + mu  # DON'T FORGET TO ADD THE MU

        img = np.reshape(img,(24,24))  # reshape flattened data into a 24*24 patc

        # We've seen the imshow method in the previous discussion :)
        ax[j].imshow( img.T , cmap="gray")
    plt.show()
```

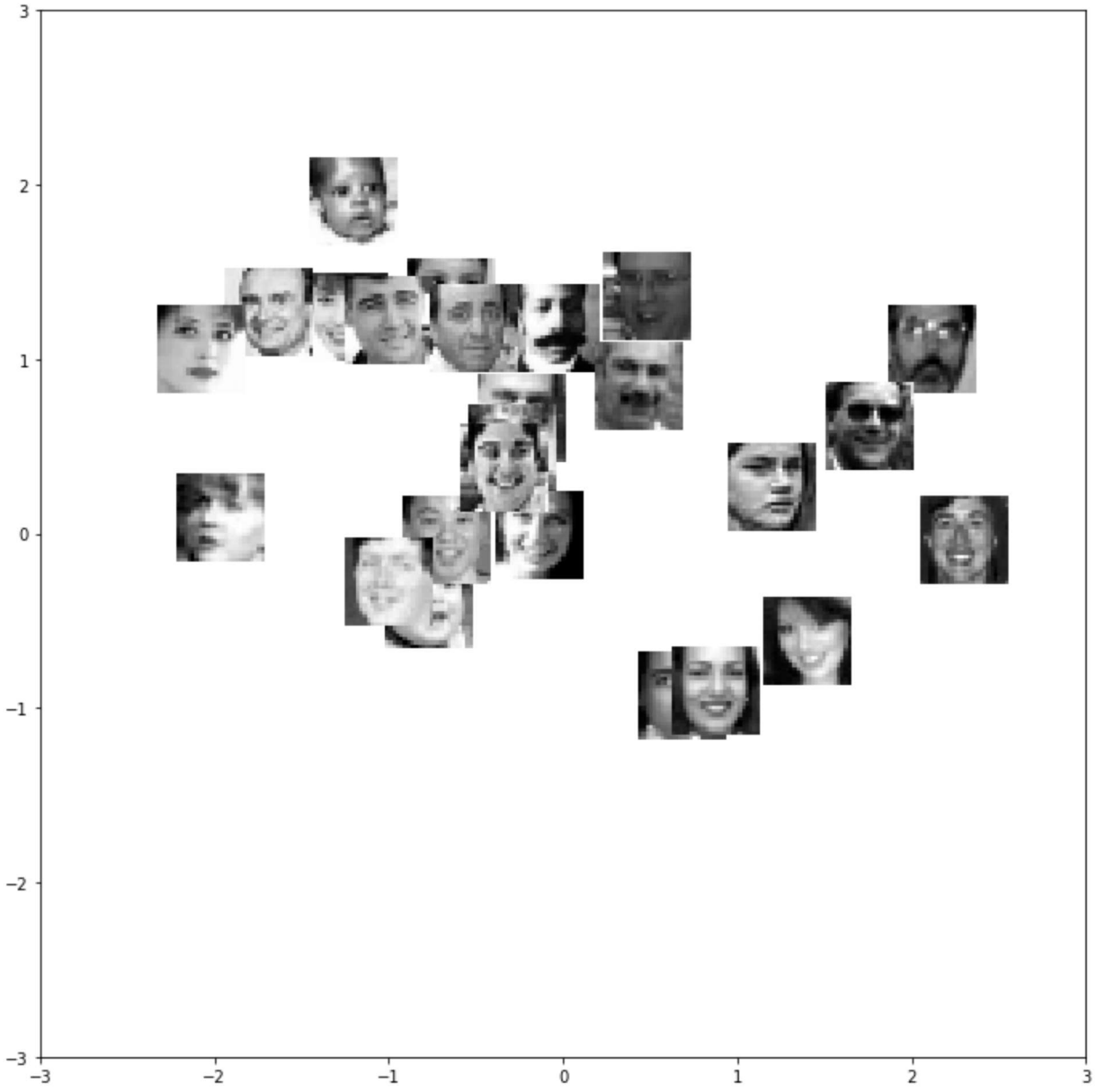




Part 6

In [46]:

```
import mltools.transforms

idx = [911, 69, 42, 24, 1, 777, 1111, 2, 12, 2000, 4915, 333, 999, 4444, 101, 808

coord,params = ml.transforms.rescale( W[:,0:2] )
plt.figure(figsize=(12,12)); plt.hold(True)

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) )
    plt.imshow( img.T , cmap="gray", extent=loc )

plt.axis( (-3,3,-3,3) )
plt.show()
```



Statement of Collaboration

I did not collaborate with anyone on this assignment.