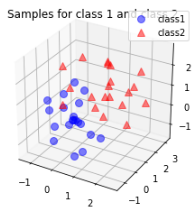
Problem 1: PCA

Question 1 & 2

In [15]:
```
import numpy as np

np.random.seed(2342) # random seed for consistency

# A reader pointed out that Python 2.7 would raise a
# "ValueError: object of too small depth for desired array".
# This can be avoided by choosing a smaller random seed, e.g. 1
# or by completely omitting this line, since I just used the random seed
 for
# consistency.

mu_vec1 = np.array([0,0,0])
cov_mat1 = np.array([[0.5,0,0],[0,0.5,0],[0,0,0.7]])
class1_sample = np.random.multivariate_normal(mu_vec1, cov_mat1, 20).T
assert class1_sample.shape == (3,20), "The matrix has not the dimensions
 3x20"

mu_vec2 = np.array([1,1,1])
cov_mat2 = np.array([[1,0,0],[0,1,0],[0,0,1]])
class2_sample = np.random.multivariate_normal(mu_vec2, cov_mat2, 20).T
assert class2_sample.shape == (3,20), "The matrix has not the dimensions
 3x20"
```

In [16]:
```
%pylab inline
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d

fig = plt.figure(figsize=(4,4))
ax = fig.add_subplot(111, projection='3d')
plt.rcParams['legend.fontsize'] = 10
ax.plot(class1_sample[0,:], class1_sample[1,:], class1_sample[2,:], 'o',
 markersize=8, color='blue', alpha=0.5, label='class1')
ax.plot(class2_sample[0,:], class2_sample[1,:], class2_sample[2,:], '^',
 markersize=8, alpha=0.5, color='red', label='class2')

plt.title('Samples for class 1 and class 2')
ax.legend(loc='upper right')

plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



Question 3 :

In [17]:
```
#Taking the whole dataset ignoring the class labels

all_samples = np.concatenate((class1_sample, class2_sample), axis=1)
assert all_samples.shape == (3,40), "The matrix has not the dimensions 3
x40"

all_samples
```

Out[17]:
```
array([[-0.57803638, -0.13543155, -1.28484711,  0.74920655,  1.2869534
4,
        -0.54386842, -0.38026263, -0.82355086,  0.25894617, -0.0535005
,
        -0.01775924, -0.01223636,  0.51089099,  0.3954618 , -0.3903928
8,
        -0.0075485 ,  1.21008454,  1.14499878,  0.08128561, -0.2135075
6,
         0.48249026,  0.35810061,  0.96735979,  1.96496396,  1.6479174
,
         2.64806311,  2.10348562,  0.23671994,  0.55785456,  0.8614741
,
         0.04565541,  1.48916082,  1.56784735,  1.69162299,  0.9471023
3,
         2.00807061,  1.1422335 ,  0.2671438 , -1.02312085,  0.1808284
5],
       [-0.51068981,  0.5675634 ,  0.2672188 , -0.26241971,  0.0860380
7,
         1.13256349,  0.00492771, -0.41352738, -0.32527264, -1.3758168
4,
         0.58268814,  0.18702968, -0.7572345 ,  0.23044568,  0.0979446
6,
         0.35748148, -0.64144668, -1.39261556, -0.37500819,  1.1121835
8,
         2.36142095,  2.82340104,  1.02071546,  0.68842849,  3.1548350
4,
         1.46531229,  0.80515182,  0.54477498,  2.48645363,  0.1036208
1,
        -0.78025375,  0.59397196,  0.26957556,  2.38907252,  1.3857337
3,
         1.17403014,  0.82615205,  1.65752048,  1.19173851, -0.4129570
4],
       [-1.05194277,  1.55148651,  0.42261453,  1.74415568, -0.6070643
1,
        -0.78749178,  0.46349746,  0.36382825,  0.36396034,  0.2859665
5,
        -0.44463939, -0.14361708, -1.32581268, -1.64653441,  1.0863276
7,
        -0.22751489, -0.08409492,  0.789675  , -0.06559604,  0.2429765
9,
         1.88096838, -0.06106761, -0.03830072,  0.99795412,  1.4771832
7,
         0.94479067, -0.01381845,  2.8483927 ,  1.77058724,  1.3225127
4,
         2.93764466,  2.05656787,  1.98396343,  1.05785065,  2.4761569
,
         1.40473695,  1.07004831, -0.11920616, -0.14114359,  1.7804703
2]])
```

In [18]:
```
mean_x = np.mean(all_samples[0,:])
mean_y = np.mean(all_samples[1,:])
mean_z = np.mean(all_samples[2,:])

mean_vector = np.array([[mean_x],[mean_y],[mean_z]])

# mean_vector = []
# mean_vector.append(mean_x)
# mean_vector.append(mean_y)
# mean_vector.append(mean_z)

# print('Mean Vector:\n', mean_vector)
```

In [19]:
```
scatter_matrix = np.zeros((3,3))
for i in range(all_samples.shape[1]):
    scatter_matrix += (all_samples[:,i].reshape(3,1) - mean_vector).dot
((all_samples[:,i].reshape(3,1) - mean_vector).T)
print('Scatter Matrix:\n', scatter_matrix)
```

```
Scatter Matrix:
 [[32.7644825  10.54811705 10.6787469 ]
 [10.54811705 45.95770145  8.15410551]
 [10.6787469   8.15410551 48.23274663]]
```

In [20]:
```
print('Covariance Matrix:\n')
covariance_matrix = []
for x in scatter_matrix:
    items = x/39
    print(items)
    covariance_matrix.append(items)


covariance_matrix = np.array(covariance_matrix)
```

```
Covariance Matrix:

[0.84011494 0.27046454 0.27381402]
[0.27046454 1.1784026  0.20907963]
[0.27381402 0.20907963 1.23673709]
```

Question 4

In [21]:
```
# eigenvectors and eigenvalues for the from the scatter matrix
eig_val_sc, eig_vec_sc = np.linalg.eig(scatter_matrix)

# eigenvectors and eigenvalues for the from the covariance matrix
eig_val_cov, eig_vec_cov = np.linalg.eig(covariance_matrix)

for i in range(len(eig_val_sc)):
    eigvec_sc = eig_vec_sc[:,i].reshape(1,3).T
    eigvec_cov = eig_vec_cov[:,i].reshape(1,3).T
    assert eigvec_sc.all() == eigvec_cov.all(), 'Eigenvectors are not id
entical'

    print('Eigenvector {}: \n{}'.format(i+1, eigvec_sc))
    print('Eigenvalue {} from scatter matrix: {}'.format(i+1, eig_val_sc
[i]))
    print('Eigenvalue {} from covariance matrix: {}'.format(i+1, eig_val
_cov[i]))
    print('Scaling factor: ', eig_val_sc[i]/eig_val_cov[i])
    print(40 * '-')
```

```
Eigenvector 1:
[[0.44660644]
 [0.60055311]
 [0.66323348]]
Eigenvalue 1 from scatter matrix: 62.80705679673633
Eigenvalue 1 from covariance matrix: 1.6104373537624705
Scaling factor:  38.999999999999986
----------------------------------------
Eigenvector 2:
[[ 0.89345702]
 [-0.33887597]
 [-0.29478403]]
Eigenvalue 2 from scatter matrix: 25.240418941734553
Eigenvalue 2 from covariance matrix: 0.6471902292752455
Scaling factor:  38.999999999999964
----------------------------------------
Eigenvector 3:
[[-0.04772042]
 [-0.72422306]
 [ 0.68791258]]
Eigenvalue 3 from scatter matrix: 38.907454837674855
Eigenvalue 3 from covariance matrix: 0.997627047119868
Scaling factor:  39.0
----------------------------------------
```

In [22]:
```
#Checking the eigenvector-eigenvalue calculation
for i in range(len(eig_val_sc)):
    eigv = eig_vec_sc[:,i].reshape(1,3).T
    np.testing.assert_array_almost_equal(scatter_matrix.dot(eigv), eig_v
al_sc[i] * eigv,
                                         decimal=6, err_msg='', verbose=
True)
```

In [23]:
```
#Sorting the eigenvectors by decreasing eigenvalues
for ev in eig_vec_sc:
    numpy.testing.assert_array_almost_equal(1.0, np.linalg.norm(ev))
    # instead of 'assert' because of rounding errors
```

In [24]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_val_sc[i]), eig_vec_sc[:,i]) for i in range(len
(eig_val_sc))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigen
values
for i in eig_pairs:
    print(i[0])
```

```
62.80705679673633
38.907454837674855
25.240418941734553
```

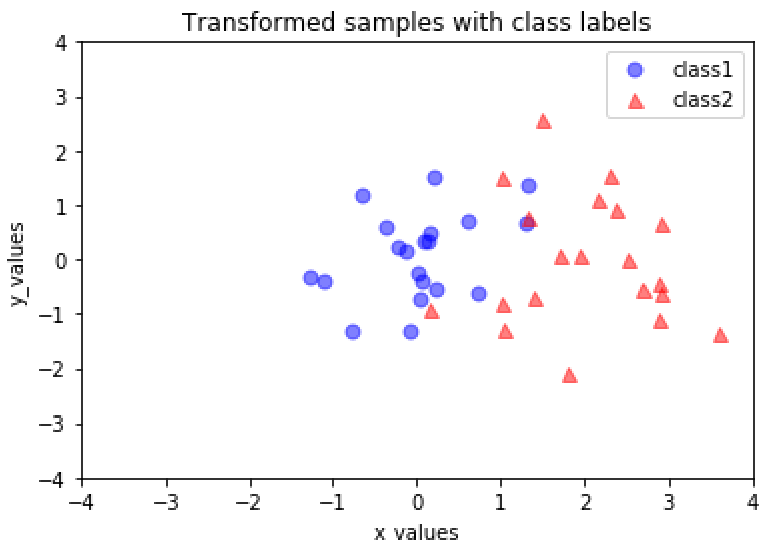
In [25]:
```
matrix_w = np.hstack((eig_pairs[0][1].reshape(3,1), eig_pairs[1][1].resh
ape(3,1)))
print('Matrix W:\n', matrix_w)
```

```
Matrix W:
 [[ 0.44660644 -0.04772042]
 [ 0.60055311 -0.72422306]
 [ 0.66323348  0.68791258]]
```

Transforming the samples onto the new subspace

In [26]:
```
transformed = matrix_w.T.dot(all_samples)
assert transformed.shape == (2,40), "The matrix is not 2x40 dimensiona
l."
```

In [27]:
```
plt.plot(transformed[0,0:20], transformed[1,0:20], 'o', markersize=7, co
lor='blue', alpha=0.5, label='class1')
plt.plot(transformed[0,20:40], transformed[1,20:40], '^', markersize=7,
color='red', alpha=0.5, label='class2')
plt.xlim([-4,4])
plt.ylim([-4,4])
plt.xlabel('x_values')
plt.ylabel('y_values')
plt.legend()
plt.title('Transformed samples with class labels')

plt.show()
```



In [ ]:

Problem 2: Low rank approximation of Mona Lisa.

In [21]:
```
from PIL import Image
import os
import sys
import numpy as np
from scipy import linalg

def perform_svd(a,rank):
    u, s, v = linalg.svd(a)
    ur = u[:, :rank]
    sr = np.matrix(linalg.diagsvd(s[:rank], rank,rank))
    vr = v[:rank, :]
    return np.asarray(ur*sr*vr)
```

In [22]:
```
def mono(filename,rank):
    path, ext = os.path.splitext(filename)
    img = Image.open(filename)
    w = img.width
    h = img.height
    gray_img = img.convert('L')
    gray_img.save(path + '_mono.jpg')
    a = np.asarray(gray_img)
    b = perform_svd(a,rank)
    img2 = Image.fromarray(np.uint8(b))
    file = path+'_r' + str(rank) + '_mono' + ext
    img2.save(file)
    print('Saved as ' + file)
```

In [23]:
```
def calculate_pixels(filename,rank):
    path, ext = os.path.splitext(filename)
    img = Image.open(filename)
    w = img.width
    h = img.height
    gray_img = img.convert('L')
    gray_img.save(path + '_mono.jpg')
    a = np.asarray(gray_img)
    b = perform_svd(a,rank)
    return b
```

In [24]:
```
def main():
    rank = 10
    argc = len(sys.argv)
    if (argc <2):
        print("usage:")
        print("$ python %s filename rank" % sys.argv[0])
        return
    filename = sys.argv[1]
    if (os.path.exists(filename) == False):
        print ("File is not found: %s" % filename)
        return

    if (argc >2):
        rank = int(sys.argv[2])

    mono(filename,rank)

    result = calculate_pixels(filename,rank)
    f = open("pixel.txt", "w")
    f.write(result)
    f.close



if __name__ == '__main__':
    main()
```

```
File is not found: -f
```

In [ ]:

$k = 5$

$(603k + 400k + k)$

1 pixel = 2 bytes

1 byte = 8 bits

$(3015 + 2000 + 5) \times 16$

$= 80,320$ bits

$k = 10$

$(603k + 400k + k)$

$(6030 + 4000 + 10) \times 16$

$= 160640$ bits

$k = 2$

$(603k + 400k + k) \times 16$

$(1206 + 800 + 2) \times 16$

$= 2008 \times 16$

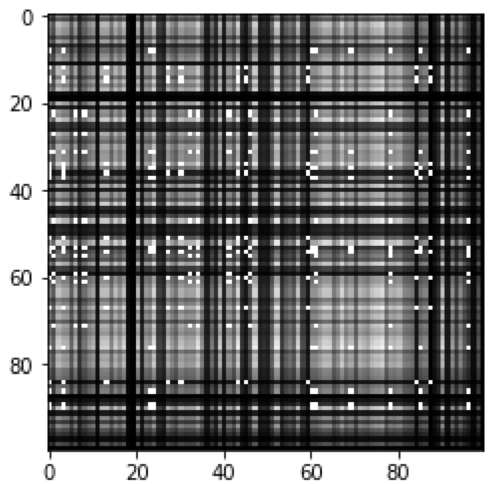
$= 32128$ bits

Problem 3: Using Low Rank Structure for Corrupted Entries.

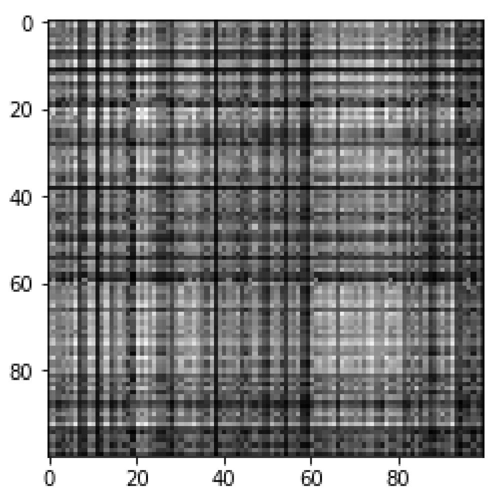
In [35]:
```
import pandas as pd
from matplotlib import pylab
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
```

In [28]:
```
first_matrix = pd.read_csv('CorrMat1.csv', header = None)
second_matrix = pd.read_csv('CorrMat3.csv', header = None)
```

In [30]:
```
pylab.imshow(first_matrix, cmap=pylab.cm.gray)
pylab.draw()
```



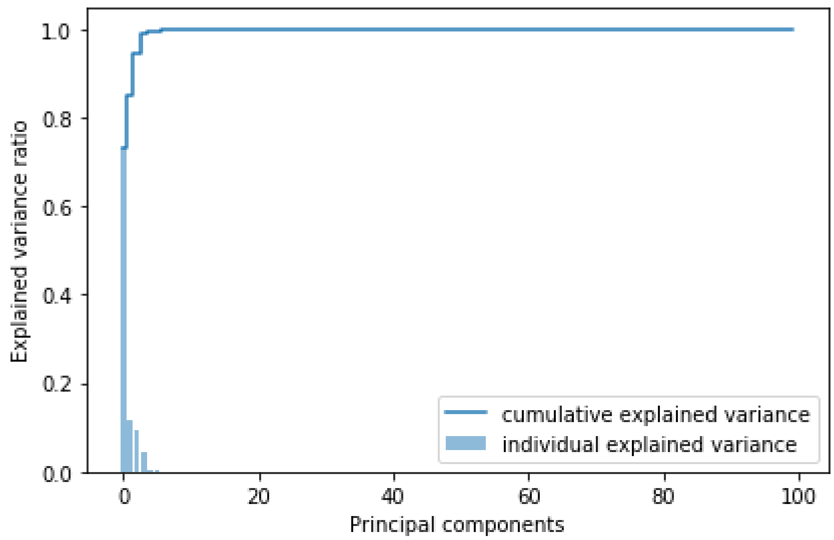
In [31]:
```
pylab.imshow(second_matrix, cmap=pylab.cm.gray)
pylab.draw()
```



In [38]:
```
covariance_matrix_first = np.cov(first_matrix.T)
eigen_values, eigen_vectors = np.linalg.eigh(covariance_matrix_first) #
 note: eigh and not eig

print('\nEigenvalues \n%s' % eigen_values)
```

```
Eigenvalues
[-3.24944244e-08 -2.75946979e-08 -2.51952861e-08 -2.13361927e-08
 -2.00190493e-08 -1.70581066e-08 -1.65031348e-08 -1.50945425e-08
 -1.45413195e-08 -1.27235548e-08 -1.19639704e-08 -9.54132031e-09
 -8.11197621e-09 -7.06903812e-09 -6.89664313e-09 -6.37221288e-09
 -5.28820263e-09 -4.69848596e-09 -4.60719230e-09 -3.08316147e-09
 -2.62790752e-09 -1.84774238e-09 -1.80143880e-09 -1.30057016e-09
 -1.17530773e-09 -7.73602771e-10 -5.34900933e-10 -4.94360294e-10
 -4.65152369e-10 -4.22991868e-10 -3.22340181e-10 -2.76135663e-10
 -2.12188500e-10 -1.81816019e-10 -1.53160980e-10 -1.19976044e-10
 -1.15309433e-10 -8.91349300e-11 -7.41460979e-11 -5.05725825e-11
 -4.99309464e-11 -3.10203034e-11 -2.96775094e-11 -1.89886869e-11
 -1.18535937e-11 -4.39986456e-12 -3.87837110e-13 -2.19872439e-25
  2.92172651e-13  4.08253638e-12  7.89671012e-12  1.00351016e-11
  1.97806693e-11  2.40588440e-11  3.54677701e-11  5.01892166e-11
  6.12910157e-11  9.49699437e-11  1.01061913e-10  1.43193796e-10
  1.99745900e-10  2.48645218e-10  3.20298393e-10  4.41278797e-10
  6.70584926e-10  7.11742675e-10  8.78717674e-10  9.26946459e-10
  1.42411878e-09  2.06045833e-09  2.15690915e-09  3.24732183e-09
  3.36126633e-09  3.66829439e-09  4.34504717e-09  4.89304925e-09
  5.36453219e-09  6.97243976e-09  8.89320425e-09  9.80884183e-09
  1.01005697e-08  1.27646782e-08  1.45148353e-08  1.65643439e-08
  1.85456240e-08  1.99624274e-08  2.00274008e-08  2.47050823e-08
  2.69804263e-08  3.41914834e-08  8.81981602e+03  1.75714419e+05
  3.74875504e+05  5.01849756e+05  1.56429836e+06  1.66149747e+06
  1.92726778e+07  3.93732876e+07  4.88452598e+07  3.06169659e+08]
```

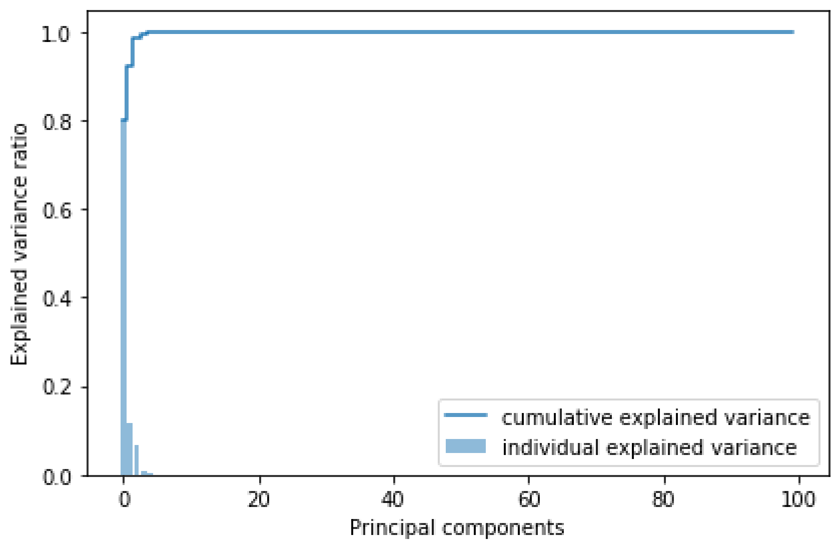
In [39]:
```
total = sum(eigen_values)
new_variance = [(i / total) for i in sorted(eigen_values, reverse=True)]
cumulative_variance = np.cumsum(new_variance)


plt.bar(range(100), new_variance, alpha=0.5, align='center', label='indi
vidual explained variance')
plt.step(range(100), cumulative_variance, where='mid', label='cumulative
 explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



In [40]:
```
covariance_matrix_second = np.cov(second_matrix.T)
eigen_values_second, eigen_vectors_second = np.linalg.eigh(covariance_ma
trix_second) # note: eigh and not eig

print('\nEigenvalues \n%s' % eigen_vectors_second)
```

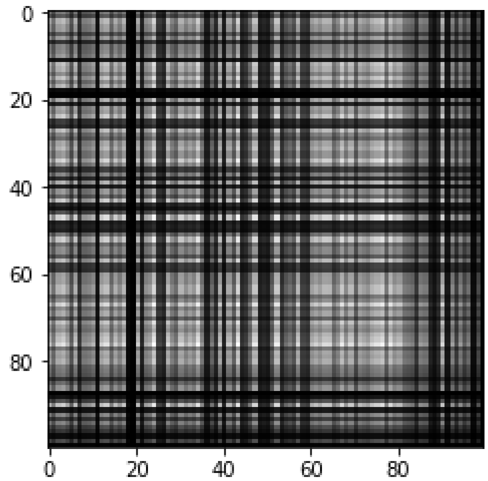
```
Eigenvalues
[[ 0.09016123 -0.02277187 -0.03349276 ...  0.00150313  0.07821712
  -0.11033289]
 [-0.46668944 -0.107717   -0.05835916 ... -0.09626476  0.12524003
  -0.10860647]
 [-0.06162484 -0.03541607  0.19708636 ...  0.16550835 -0.00565148
  -0.09178963]
 ...
 [-0.01423249  0.05621942  0.04830045 ... -0.18332952 -0.07502791
  -0.06516529]
 [ 0.05040643  0.04467897  0.09339429 ...  0.0053285   0.13795525
  -0.0756077 ]
 [ 0.03332434  0.01739252  0.02933882 ... -0.06101048  0.11606164
  -0.03700899]]
```

In [41]:
```
total_second = sum(eigen_values_second)
new_variance_second = [(i / total_second) for i in sorted(eigen_values_s
econd, reverse=True)]
cumulative_variance_second = np.cumsum(new_variance_second)

plt.bar(range(100), new_variance_second, alpha=0.5, align='center', labe
l='individual explained variance')
plt.step(range(100), cumulative_variance_second, where='mid', label='cum
ulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('./figures/pca1.png', dpi=300)
plt.show()
```

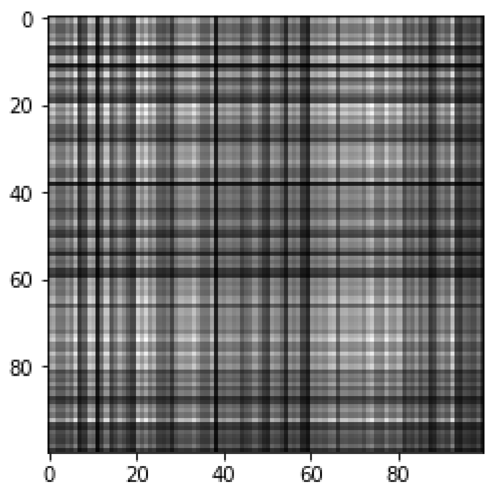


In [42]:
```
def low_rank_approx(arr, r):
    """
    Computes an r-rank approximation of a matrix
    given the component u, s, and v of it's SVD
    Requires: numpy
    """

    SVD = np.linalg.svd(arr, full_matrices=False)
    u, s, v = SVD
    Ar = np.zeros((len(u), len(v)))
    for i in range(r):
        Ar += s[i] * np.outer(u.T[i], v[i])
    return Ar
```

In [44]:
```
#LOW Rank Approximation with k=4
import pylab
approx_first_matrix = low_rank_approx(first_matrix,1)
pylab.imshow(approx_first_matrix, cmap=pylab.cm.gray)
pylab.draw()
```



In [45]:
```
approx_second_matrix = low_rank_approx(second_matrix,1)
pylab.imshow(approx_second_matrix, cmap=pylab.cm.gray)
pylab.draw()
```



In [ ]:

Answer : **Out of 10 women with a positive mammogram, about 1 has breast cancer;**

BAYES THEOREM

Q Based on medical statistics, 10 out of every 1000 women have breast cancer. Of these 10 women with breast cancer, 9 test positive. Of the 9900 women without cancer, about 89 nevertheless test positive. A woman tests positive and wants to know whether she has breast cancer for sure or at least what the chances are

What is the best answer?

Ans :- Event C → patient has cancer

$$P\left(\frac{\text{women}}{\text{cancer}} \Big| +ve\right) = ?$$

$$P(C) = \frac{10}{1000} = 0.001$$

$$P(TP|C) = \frac{9}{10} = 0.9$$

4. $$P\left(+ve \Big| \frac{\text{women}}{\text{cancer}}\right) = \frac{9}{10} = 0.9$$

1. $$P(\text{cancer}) - 0.01$$

2. $$P(\text{not cancer}) = 0.99$$

3. $$P\left(+ve \Big| \frac{\text{women}}{\text{no cancer}}\right) = \frac{89}{990} = 0.08$$

5. $P(\text{women cancer} \mid +ve)$

$$= \frac{P(\text{women cancer}) \cdot P(+ve \mid \text{cancer})}{P(+ve)}$$

$$= \frac{0.01 * 0.9}{P(+ve)} \rightarrow \text{Equation 1}$$

$$\hookrightarrow P(\text{cancer}) * P(+ve \mid \text{cancer}) + P(\text{not cancer}) * P(+ve \mid \text{not cancer})$$

$$= (0.01 * 0.9) + (0.99 * 0.08)$$
$$= 0.009 + 0.08$$
$$= 0.08900$$

Substituting in equation ①,

$$\frac{0.01 * 0.9}{0.08900}$$

$$= 0.10$$

The probability of a woman having cancer given a test positive is 10%.