1 Problem 1

Prove that the set of first principal components has maximum variance among any other set of principal components.

Consider a data set $X = \{x^{(\mu)} \mid \mu \in [1, P]\}$ in an N dimensional feature space. Let $C = XX^T$ be the covariance of X, and $\phi^{(i)}$ the eigenvectors of the linear equation $C\phi^{(i)} = \lambda_i \phi^{(i)}$ with eigenvectors λ_i for $i \in [1, N]$. Furthermore, order $(\lambda_i, \phi^{(i)})$ such that for $i > k$ we have $\lambda_i < \lambda_k$. Furthermore let $a_i^{(\mu)}$ be the i-th principal component of $x^{(\mu)}$ given by $a_i^{(\mu)} = \langle x^{(\mu)}, \phi^{(i)} \rangle$.

$a_i^{(\mu)}$ is thus a projection of $x^{(\mu)}$ onto the eigenvector $\phi^{(i)}$. The variance of a_i is thus the variance of X with respect to the i-th eigenvector. Since the eigenvectors $\{\phi^{(i)}\}$ are of the covariance of X, the covariance of X with respect to any $\phi^{(i)}$ is a diagonal matrix and is equal to the variance of X with respect to the same eigenvector. Therefore, $\mathrm{Var}(a_i) = \mathrm{Cov}(X)_{\phi^{(i)}} = \lambda_i$. By hypothesis, $\lambda_1 > \lambda_i \, \forall i > 1$. Therefore $\mathrm{Var}(a)$ is maximal along the first principal component.

Heuristically, the first principal component is constructed to be the component along which the covariance of X is maximal. Therefore, the projection of the data set X onto the different components, which is denoted by a_i, has maximal variance along the first principal component, since the variance of X along the component is equal to the covariance with respect to that component.

2 Problem 2

Calculate and display eigenfaces as well as approximation faces.

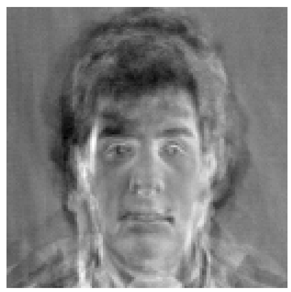
Using the script below and `eigenface.py`, I performed PCA on the set of faces supplied. Figure 1 shows the average face, and Figure 2 show the first eight eigenfaces (in order) determined from the data set. Figure 3 shows the first face given in the set, and Figure 4 shows the first eight approximation eigenfaces.

`problem2.py`

```python
#!/usr/local/bin/ipython
# Athanasios Athanassiadis March 2012
from eigenface import *

infolder = 'faces'
outfolder = 'eigenfaces'
outfolder2 = 'face_approx'

print 'loading faces'
origfaces, faces, shape = load_faces(infolder)
avgface = origfaces.sum(0) / origfaces.shape[0]

print 'computing eigenfaces'
eigfaces = eigenfaces(faces, 8)

print 'saving output'
imsave(os.path.join(outfolder, 'avgface.png'),
       avgface)

for i in range(len(eigfaces)):
    imsave(os.path.join(outfolder,'eigenface%d.png' % i),
           eigfaces[i].reshape(shape))


print 'calculating approximation faces for first face'
face = origfaces[0]

fapproxes = approxiface(face.flatten(), eigfaces, avgface.flatten(), 8)

print 'saving output'
imsave(os.path.join(outfolder2, 'original.png'),
       face)

for i in range(len(fapproxes)):
    imsave(os.path.join(outfolder2, 'approx%d.png' % (i+1)),
           fapproxes[i].reshape(shape))
```

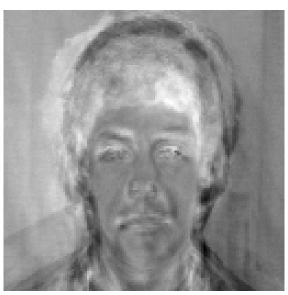
Figure 1: Average Face

(a)



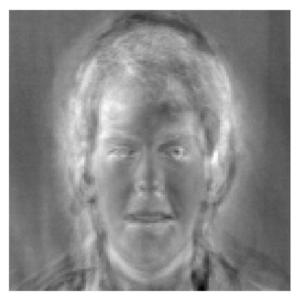
(b)

(c)



(d)

(e)



(f)

(g)

(h)

Figure 2: First 8 eigenfaces

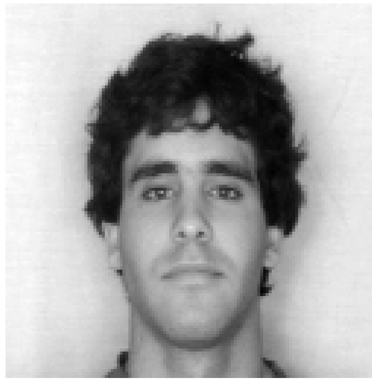
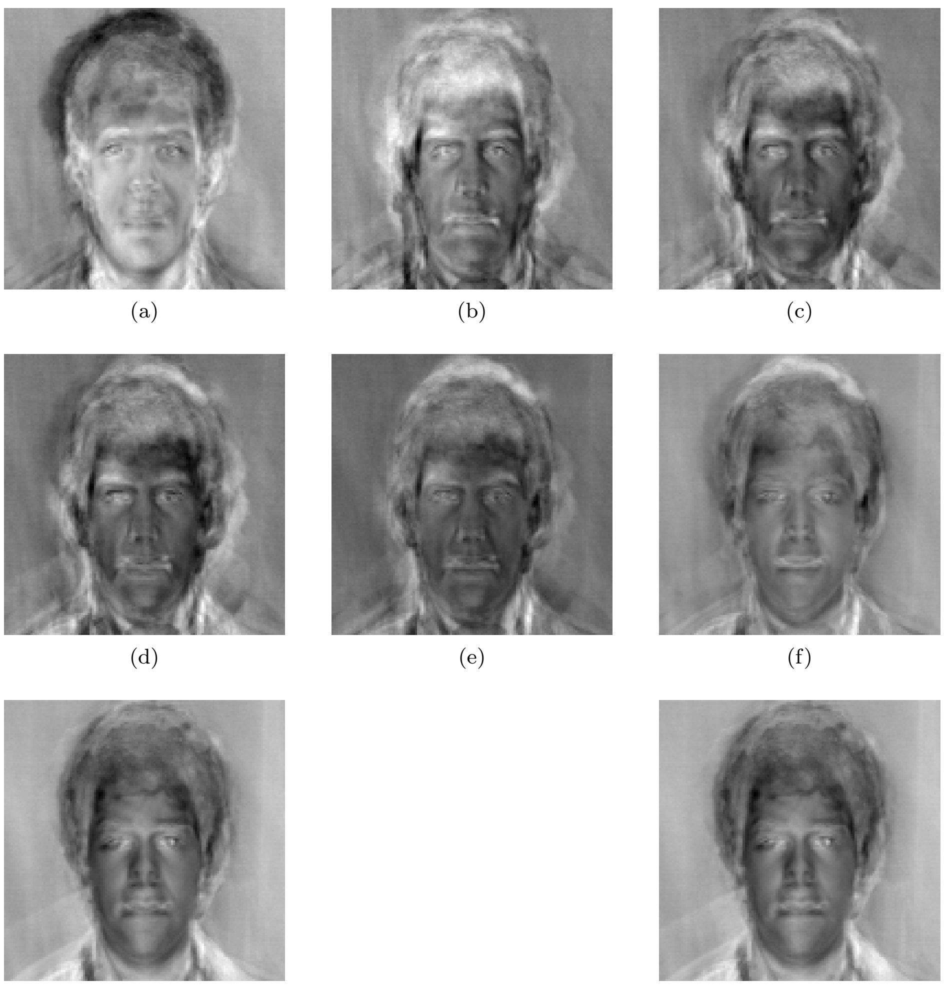


Figure 3: Input Face



(a) (b) (c)

(d) (e) (f)

(g) (h)

Figure 4: First 8 approximation faces

3 Appendix: Common Code

The base code used for Problem 1 in this homework is contained in `eigenface.py`.

`eigenface.py`

```
# Athanasios Athanassiadis March 2012
import os
import numpy as np
from numpy.linalg import eig
from scipy.misc import imread, imsave


cov = np.cov

def load_faces(folder):
    '''
    load_faces
            load a set of faces from a folder of tif images
            and return reduced set of faces with shape info
    '''

    filelist = os.listdir(folder)

    origfaces = []
    for fn in filelist:
        origfaces.append(
            imread(os.path.join(folder,fn)))

    origfaces = np.array(origfaces).astype(np.int16)

    avgface = origfaces.sum(0) / origfaces.shape[0]
    shape = avgface.shape

    # make faces to process zero-mean
    faces = np.array([(face-avgface).flatten() for face in origfaces]).T


    return origfaces, faces, shape

def eigenfaces(faces, n=None):
    '''
    eigenfaces
        returns the first n eigenfaces (principal components)
        calculated from an array of face which has been reduced to 2D

        eigenfaces are the eigenvectors of the covariance matrix of the
        faces matrix

        this computation is simplified using the Turk-Pentland Trick

    '''

    C = cov(faces.T, bias=1)

    # columns in eigvecs represent individual eigenvectors
    # that correspond to each eigenval
```

```
    eigvals, eigvecs = eig(C)

    if n is None or n>len(eigvals):
        n = len(eigvals)

    # sort by eigenvalue (decreasing)
    # and transpose so rows contain eigenvectors
    eigzip = zip(eigvals, eigvecs.T)
    eigzip.sort(key = lambda x: x[0])
    eigenvecs_sorted = np.array([pair[1] for pair in eigzip[::-1]])

    eigfaces = np.zeros((n, faces.shape[0]))

    # calculate eigenfaces from eigenvectors and zero-mean faces
    # according to Turk-Pentland
    for l in range(n):
        for k in range(faces.shape[1]):
            eigfaces[l] += (eigvecs[l,k] * faces[:,k])

    return eigfaces.astype(np.int16)

def approxiface(face, eigfaces, avgface, n=None):
    '''
    approxiface
        approximate a face b,ased on eigfaces
        return first n approximations

    '''

    if n is None or n>len(eigfaces):
        n = len(eigfaces)

    weights = np.zeros(n)

    # calculate projection on to each axis (eigenface)
    for k in range(n):
        weights[k] = np.dot(eigfaces[k], face-avgface)

    # normalize so that norm(weights) = 1
    weights /= np.sqrt((weights**2).sum())

    # calculate first n approximation faces
    approxes = np.zeros((n, eigfaces.shape[1]))
    approxes[0] = weights[0] * eigfaces[0]
    for k in range(1,n):
        approxes[k] = approxes[k-1] + (weights[k] * eigfaces[k])

    return approxes
```