**Student Names and IDs**

- Yiteng Lu (2488152)
- Zengtian Deng (2207324)
- Wenge Xie (2466824)

## Homework 3

# Part 1: Singular Values/Vectors and Eigenvalues/Vectors

## Problem 1.1 (ES)

Find exact expressions (as opposed to approximate numerical values) for the eigenvalues $\lambda_1$ , $\lambda_2$ and corresponding *unit-norm* eigenvectors $\mathbf{e}_1$ , $\mathbf{e}_2$ of the matrix

$$C = \begin{bmatrix} 0 & 1 \\ -2 & -2 \end{bmatrix}$$

Use the symbol $i$ for the imaginary unit, if needed. When normalizing, recall that the squared norm of a complex vector $\mathbf{a}$ is the inned product of the vector with itself, that is, $\mathbf{a}^*\mathbf{a}$, where $\mathbf{a}^*$ is the transpose and conjugate of $\mathbf{a}$.

No need to show your derivation, unless you want partial credit should your answer be wrong.

## Answer

$$\begin{bmatrix} \lambda_1 = -1 + i \\ \lambda_2 = -1 - i \end{bmatrix}, \quad \mathbf{e_1} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{-1+i}{\sqrt{3}} \end{bmatrix} \quad \mathbf{e_2} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{-1-i}{\sqrt{3}} \end{bmatrix}$$

## Problem 1.2

Find approximate numerical expressions, to four decimal digits after the period, for the left singular vectors $\mathbf{u}_1$ , $\mathbf{u}_2$ , singular values $\sigma_1$ , $\sigma_2$ , and right singular vectors $\mathbf{v}_1$ , $\mathbf{v}_2$ of the matrix $C$ in Problem 1.1.

## Answer

$$\mathbf{u_1} = \begin{bmatrix} -0.2567 \\ 0.9665 \end{bmatrix} \quad \mathbf{u_2} = \begin{bmatrix} 0.9665 \\ 0.2567 \end{bmatrix}$$

$$\sigma_1 = 2.9206 \quad \sigma_2 = 0.6847$$

$$\mathbf{v_1} = \begin{bmatrix} -0.6618 \\ -0.7497 \end{bmatrix} \quad \mathbf{v_2} = \begin{bmatrix} -0.7497 \\ 0.6618 \end{bmatrix}$$

## Problem 1.3 (ES)

Use the equation that defines eigenvalues $\lambda$ and eigenvectors $\mathbf{e}$ of a square matrix $A$ to show that if $A$ is symmetric and real then its eigenvalues are real, and its eigenvectors can be scaled to be real.

Hint: Write the equation and its Hermitian (*i.e.*, its transpose-conjugate):

$$A\mathbf{e} = \lambda\mathbf{e}$$
$$\mathbf{e}^* A^T = \lambda^* \mathbf{e}^*$$

Then multiply the first by $\mathbf{e}^*$ from the left and the second by $\mathbf{e}$ from the right, and reason from there. Justify all your steps.

## Answer

From

$$A\mathbf{e} = \lambda\mathbf{e}$$
$$\mathbf{e}^* A^T = \lambda^*\mathbf{e}^*$$

we obtain

$$\mathbf{e}^* A\mathbf{e} = \mathbf{e}^*\lambda\mathbf{e}$$
$$\mathbf{e}^* A^T\mathbf{e} = \lambda^*\mathbf{e}^*\mathbf{e}$$

Because $\lambda$ is just a value, $\mathbf{e}^*\lambda\mathbf{e} = \lambda\mathbf{e}^*\mathbf{e}$, In addtion A is a symmetric matrix, so we have $A = A^T$.
Therefore,

$$\mathbf{e}^* A\mathbf{e} = \mathbf{e}^* A^T\mathbf{e} \quad \Rightarrow \quad \lambda\mathbf{e}^*\mathbf{e} = \lambda^*\mathbf{e}^*\mathbf{e} \quad \Rightarrow \quad \lambda = \lambda^*$$

$\lambda = \lambda^*$ indicates that $\lambda$ equals to the conjugate of itself which means $\lambda$ must be real
Given:

$$A\mathbf{e} = \lambda\mathbf{e}$$

Assmue:

$$\mathbf{e} = a + bi$$

Then we have:

$$A(a + bi) = \lambda(a + bi) \quad \Rightarrow \quad Aa + Abi = \lambda a + \lambda bi$$

Only under the condition that $\lambda$ is real (which has been proven in the first part) can we prove that $Aa = \lambda a$; 'a' is the real part of the eigenvector $\mathbf{e}$ and itself is also an eigenvector of $\lambda$ for $Aa = \lambda a$. At that point, we can claim A must have eigenvectors that can be scaled into something real just like 'a'.

## Problem 1.4 (ES)

Refer to the geometric meaning of the SVD to argue that the SVD of a tame matrix $A$ is unique up to appropriate substitutions of $U$ and $V$ with siblings of each. Be detailed and accurate in your reasoning, and specify all appropriate sibling substitutions. That is, if

$$A \;=\; [\mathbf{u}_1,\ \mathbf{u}_2]\,\Sigma\,[\mathbf{v}_1,\ \mathbf{v}_2]^T$$

is an SVD of a tame matrix $A$, what are all the other SVDs of $A$?

[Note: If $A$ were not tame, then its SVD would not necessarily be unique up to sibling substitutions of $U$ and $V$. Make sure you explain what it is that makes tame matrices special in this regard.]

## Answer

If A is a tame matrix, then other SVDs can be:

$$[-\mathbf{u}_1,\ \mathbf{u}_2]\ \Sigma\ [-\mathbf{v}_1,\ \mathbf{v}_2]^T$$
$$[\mathbf{u}_1,\ -\mathbf{u}_2]\ \Sigma\ [\mathbf{v}_1,\ -\mathbf{v}_2]^T$$
$$[-\mathbf{u}_1,\ -\mathbf{u}_2]\ \Sigma\ [-\mathbf{v}_1,\ -\mathbf{v}_2]^T$$

From geometric intuition, we can know that there must exists two orthonormal vectors $v_1, v_2$ that can span the whole row space. The loci of this two vectors is a unit circle. By multiplying with matrix $A$, it will project the unit circle into the range as a ellipsoid. $v_1, v_2$ are projected as the major and minor axis of the ellipsoid. These two axis are represented by a pair of orthonormal vectors $u_1, u_2$ By replace matrix $V$ with its sibling matrix, we are actually flipping the direction of the v-vectors. They are now pointing at the opposite direction with their norm unchanged. Under this circumstance, the two new vectors spanned the same row space. The projection of these two vectors into the range with still be the major and minor axis of the ellipsoid with the orthornormal vectors $u_1, u_2$ pointing at corresponding opposite direction. So, the new $U$ matrix will be the correspond sibling matrix of the previous $U$ matrix. These u-vectors will also span the same range because they are still have the same ellipsoid loci.

The tame definition of the matrix $A$ gives us some very good qualities. Because tame matrix has two distinct single values, it has to be full rank, which means there will not exist a null space and a left null space. By making the v-vectors or u-vectors pointing at the opposite direction will not guarantee that they can still span the same row space and range if there exist some vectors in $U$ and $V$ matrice representing the null space and left null space.

## Problem 1.5 (ES)

Show that if $A = U\Sigma V^T$ is the SVD of a symmetric, tame matrix $A$, then $U$ and $V$ are siblings of each other.

[Hint: Write the transpose of $A = U\Sigma V^T$ and reason from there, using results proven in previous problems.]

## Answer

Given:

$$A = U\Sigma V^T$$

We have:

$$A^T = V\Sigma^T U^T \;\Rightarrow\; A = V\Sigma U^T \text{ Because A is symmetrix and } \Sigma \text{ is a diagonal matrix}$$

Since we have proven in the Problem 1.4, the SVD of a tame matrix A is unique up to the siblings of $U$ and $V$, $U$ and $V$ cannot be same becuase tame matrix ensures that it has two distinct sigular values. Under that circumstance, $V$ has to be one of the siblings of U because both U and V can be the right sigular vectors in the SVD.

## Problem 1.6 (ES)

Show that the singular values of a symmetric, tame matrix $A$ are the absolute values of its eigenvalues; the left singular vectors (columns of $U$) are its eigenvectors (up to a nonzero scale factor); and the matrix $V$ of right singular vectors is a sibling of $U$.

[Hint: Write the two equations $A\mathbf{e}_1 = \lambda_1\mathbf{e}_1$ and $A\mathbf{e}_2 = \lambda_2\mathbf{e}_2$ together as a single matrix equation $AE = E\Lambda$, where

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

and reason from there, using results proven so far. Warning: Eigenvalues, even when real, can be negative.]

## Answer

$$AE = E\Lambda$$
$$AEE^T = E\Lambda E^T$$

Because E is orthgonal matrix so:

$$AEE^T = A = E\Lambda E^T$$

We also have:

$$A = U\Sigma V^T$$

Given $A^T = E\Lambda E^T$ and $A^T = V\Sigma U^T$:

$$A^T A = E\Lambda E^T E\Lambda E^T = E\Lambda^2 E^T$$
$$A^T A = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T$$

$$E\Lambda^2 E^T = V\Sigma^2 V^T$$

For the operands on both side they have the same form which is $AB^2A^T$ we could treat $E = V$ and $\Lambda^2 = \Sigma^2$ Given that for a tame matrix all the sigular values are greater than 0 so $|\Lambda| = \Sigma$.
From question 1.5, we confirmed that $V$ is a sibling of U, and $V = E$ therefore $E$ is also a sibling of $U$.
Given $U$ is $[u_1 \quad u_2]$, one of its siblings is $[-u_1, -u_2]$. Since $E$ is one of $U$'s siblings, so $E$ can be $[-u_1, -u_2]$. If we scale all the eigenvectors $e_1$ , $e_2$ with -1, then E would be equal to U. Then it is true that the left sigular vectors U of A are its eigenvectors (up to somescale).

## Problem 1.7 (ES)

Show that the left and right singular vectors of a tame, symmetric, and positive definite matrix $A$ are its (unit-norm) eigenvectors, and its singular values are its eigenvalues.

[Hint: Given problem 1.6, this is immediate.]

## Answer

Since matrix $A$ is positive definite, its eigenvalue, by definition, is positive. Therefore, one can prove that $\lambda_A = |\lambda_A|$. Problem 1.6 shows that the singular values of a tame, symmetric matrix are the absolute value of eigenvalues,which in this case is essentially the eigenvalue of the matrix $A$

Based on the provement shown in Problem 1.6, the left and right singular vectors of a tame and symmetric matrix $A$ are its eigenvectors.

### Problem 1.8 (ES)

Show that if $A$ is any real $n \times 2$ matrix with singular values $\sigma_1 > \sigma_2 > 0$ and SVD $A = U\Sigma V^T$, and $S = A^T A$, then the eigenvalues of $S$ are $\sigma_1^2$ and $\sigma_2^2$, and the corresponding eigenvectors are the columns of $V$.

[Hint: Write $S$ in terms of $U$, $\Sigma$, $V$ and recall that two eigenvalue equations,

$$B\mathbf{x}_1 = \lambda_1 \mathbf{x}_1 \quad \text{and} \quad B\mathbf{x}_1 = \lambda_1 \mathbf{x}_1$$

can be written in matrix form as

$$BX = X\Lambda \quad \text{where} \quad X = [\mathbf{x}_1, \mathbf{x}_2] \quad \text{and} \quad \Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}.$$

Tameness is not needed here.]

### Answer

$$S = A^T A = V\Sigma^2 V^T$$
$$SV = V\Sigma^2 V^T V = V\Sigma^2$$
$$S[\mathbf{v_1}\ \mathbf{v_2}] = [\mathbf{v_1}\ \mathbf{v_2}] \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} = [\mathbf{v_1}\sigma_1^2\ \ \mathbf{v_2}\sigma_2^2]$$

thus we have:

$$S\mathbf{v_1} = \sigma_1^2 \mathbf{v_1}$$
$$S\mathbf{v_2} = \sigma_2^2 \mathbf{v_2}$$

The two above equations follow the canonical form of eigenvectors and eigenvalues, so the columns of V are the eigenvectors of S and the corresponding eigenvalues are $\sigma_1^2$ and $\sigma_2^2$.

## Part 2: Orthogonality of the Right Singular Vectors

Use calculus to prove that $\mathbf{v}_1$ and $\mathbf{v}_2$ as defined above are orthogonal to each other. This implies that the matrix $V$ in the SVD $A = U\Sigma V^T$ of any $n \times 2$ matrix $A$ is orthogonal.

## Answer

The way to find $v_1$ and $v_2$ is to minimize or maximize $||Ax^2||$. Choose $\mathbf{x}$ to be $[cos(\theta), sin(\theta)]^T$. Let

$$f(\theta) = ||Ax||^2$$

then,

$$\begin{aligned}\frac{f(\theta)}{dx} &= \frac{d(||Ax(\theta)||^2)}{d\theta} \\ &= [(x^T(\theta)A^T) \cdot (Ax(\theta))]'\end{aligned}$$

Because $A = \begin{bmatrix} a_{11}\, a_{12} \\ a_{21}\, a_{22} \\ \cdots \\ a_{n1}\, a_{n2} \end{bmatrix}, A_T = \begin{bmatrix} a_{11}\, a_{12}\, \cdots\, a_{n1} \\ a_{12}\, a_{22}\, \cdots\, a_{n2} \end{bmatrix}, x = \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix}, x^T = \begin{bmatrix} cos(\theta) & sin(\theta) \end{bmatrix},$

then,

$$
\begin{aligned}
\frac{f(\theta)}{dx} &= [x^T(\theta)(A^TA)x(\theta))]' \\
&= \left( \begin{bmatrix} cos(\theta) & sin(\theta) \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n a_{i1}^2 & \sum_{i=1}^n a_{i1}a_{i2} \\ \sum_{i=1}^n a_{i1}a_{i2} & \sum_{i=1}^n a_{i2}^2 \end{bmatrix} \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} \right)' \\
&= \left( \begin{bmatrix} \left( \sum_{i=1}^n a_{i1}^2 \right)cos(\theta) + \left( \sum_{i=1}^n a_{i1}a_{i2} \right)sin(\theta) & \left( \sum_{i=1}^n a_{i1}a_{i2} \right)cos(\theta) + \left( \sum_{i=1}^n a_{i2}^2 \right)sin(\theta) \end{bmatrix} \begin{bmatrix} c \\ s \end{bmatrix} \right. \\
&= \left( \left( \sum_{i=1}^n a_{i1}^2 \right)cos^2(\theta) + \left( \sum_{i=1}^n a_{i2}^2 \right)sin^2(\theta) + 2\left( \sum_{i=1}^n a_{i1}a_{i2} \right)sin(\theta)cos(\theta) \right)' \\
&= \left( \sum_{i=1}^n a_{i1}^2 - \left( \sum_{i=1}^n a_{i1}^2 \right)sin^2(\theta) + \left( \sum_{i=1}^n a_{i2}^2 \right)sin^2(\theta) + 2\left( \sum_{i=1}^n a_{i2}a_{i2} \right)sin(\theta)cos(\theta) \right)' \\
&= \left( \sum_{i=1}^n a_{i1}^2 - \left( \sum_{i=1}^n a_{i2}^2 - \sum_{i=1}^n a_{i1}^2 \right)sin^2(\theta) + \left( \sum_{i=1}^n a_{i1}a_{i2} \right)sin(2\theta) \right)' \\
&= \left( \sum_{i=1}^n a_{i1}^2 - \left( \sum_{i=1}^n a_{i2}^2 - \sum_{i=1}^n a_{i1}^2 \right)\frac{1-cos(\theta)}{2} + \left( \sum_{i=1}^n a_{i1}a_{i2} \right)sin(2\theta) \right)' \\
&= \left( \sum_{i=1}^n a_{i1}^2 - \frac{\sum_{i=1}^n (a_{i2}^2 - a_{i1}^2)}{2} - \frac{\sum_{i=1}^n (a_{i2}^2 - a_{i1}^2)}{2}cos(2\theta) + \left( \sum_{i=1}^n a_{i1}a_{i2} \right)sin(2\theta) \right)' \\
&= \left( \sum_{i=1}^n a_{i1}^2 - \frac{\sum_{i=1}^n (a_{i2}^2 - a_{i1}^2)}{2} + \sqrt{\frac{\left( \sum_{i=1}^n (a_{i2}^2 - a_{i1}^2) \right)^2}{4} + \left( \sum_{i=1}^n a_{i1}a_{i2} \right)^2}\ \ cos(2\theta - \phi) \right)' \\
&= -\sqrt{\left( \sum_{i=1}^n (a_{i2}^2 - a_{i1}^2) \right)^2 + 4\left( \sum_{i=1}^n a_{i1}a_{i2} \right)^2}\ \ sin(2\theta - \phi)
\end{aligned}
$$

where,

$$\phi = \frac{\sum_{i=1}^n (a_{i2}^2 - a_{i1}^2)}{2\sum_{i=1}^n a_{i1}a_{i2}}$$

Let $\frac{f(\theta)}{dx} = 0$, then,

$$sin(2\theta - \phi) = 0$$

There exists two posible cases:

$$\begin{cases} 2\theta_1 - \phi = 0 \\ 2\theta_2 - \phi = \pi \end{cases}$$

By solving the equations, we can get

$$\begin{cases} \theta_1 = \frac{\phi}{2} \\ \theta_2 = \frac{\phi}{2} + \frac{\pi}{2} \end{cases}$$

So,

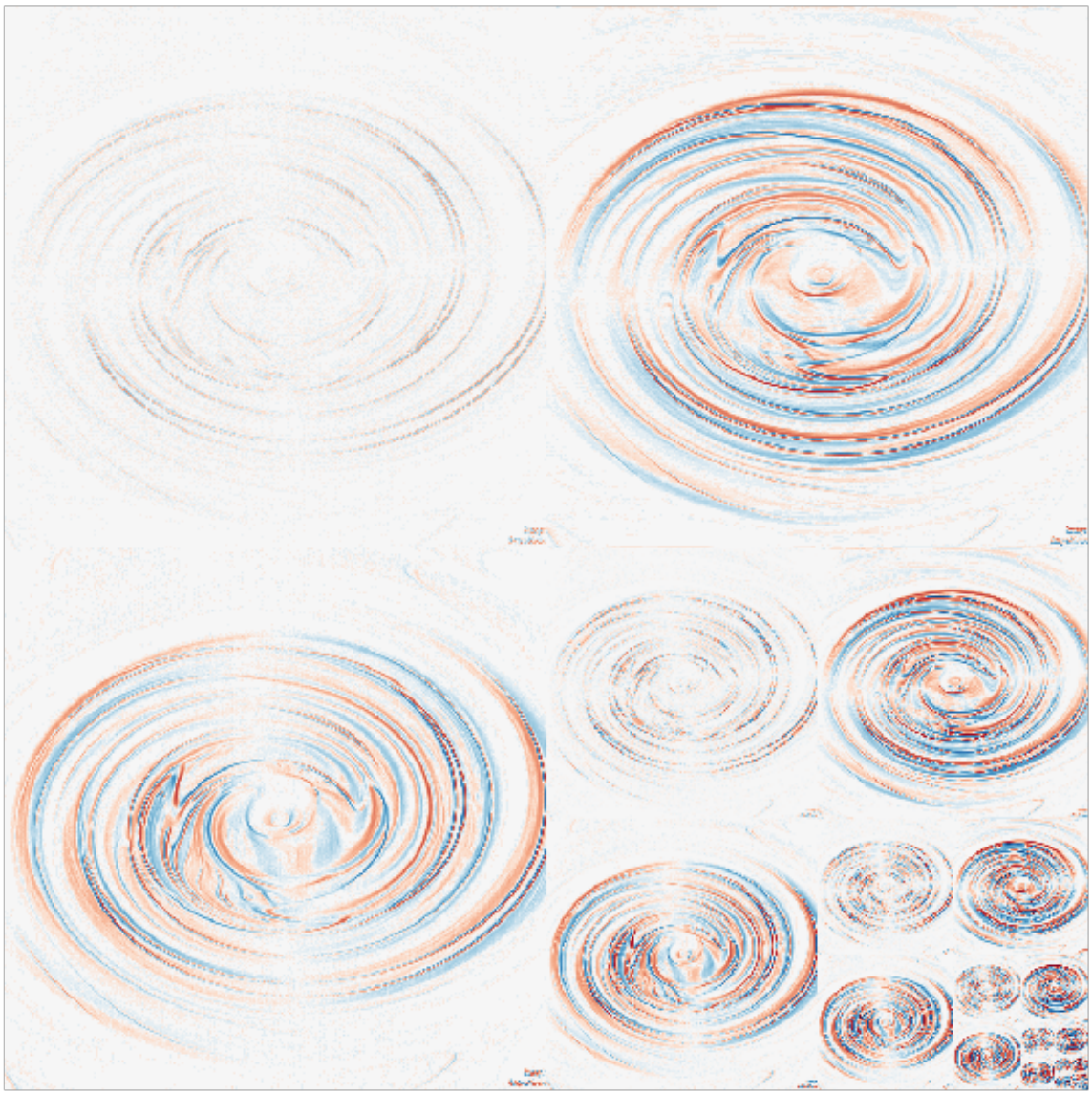
$$\theta_1 - \theta_2 = \frac{\pi}{2}$$

Because $\frac{f(\theta)}{dx} = 0$, so $\theta_1$ and $\theta_2$ are corresponding to the angle of the major and minor axis of the ellipsoid. Their difference is $\frac{\pi}{2}$ indicates that they are orthogonal to each other.

# Part 3: Haar Pyramids

In [1]:
```
import numpy as np
from skimage import io
from matplotlib import pyplot as plt
%matplotlib inline

def i2p(img):
    img = np.ndarray.astype(img, 'float')
    sz = img.shape
    depth = np.rint(np.log2(sz))[0].astype(int)
    assert np.all(2 ** depth == sz[:2]), 'Image must be square with 2^
k rows'
    E = np.array([[1, -1, 1, -1], [1, 1, -1, -1],
                  [1, -1, -1, 1], [1, 1, 1, 1]]) / 4;
    detail = []
    for level in range(depth):
        Q = [img[::2, ::2], img[1::2, ::2], img[::2, 1::2], img[1::2,
1::2]]
        det = []
        for d in range(4):
            det.append(np.zeros_like(Q[0]))
            for q in range(4):
                det[d] += E[d, q] * Q[q]
        detail.append(det[:3])
        img = det[3]
    return detail, img

def composite(detail, mean):
    n = detail[0][0].shape[0]
    n2 = n * 2
    comp = np.full((n2, n2), mean[0, 0])
    a, b, c = 0, n, n2
    for d in detail:
        comp[a:b, b:c] = d[0]
        comp[b:c, a:b] = d[1]
        comp[a:b, a:b] = d[2]
        a, n = b, int(n/2)
        b += n
```

```
    return comp

def show(img, size=(5, 5), cmap='gray'):
    plt.figure(figsize=size)
    plt.imshow(img, cmap=cmap)
    plt.axis('off')
    plt.show()

img = io.imread('ripples.png')
show(img)

(detail, average) = i2p(img)
comp = composite(detail, average)
recolor = np.multiply(np.log(1 + np.abs(comp)), np.sign(comp))
show(recolor, (10, 10), 'RdBu')
```



## Problem 3.1

After carefully studying the code provided, and in particular how equation (∗) is implemented in function `i2p`, implement a function with header

```
def p2i(detail, average):
```

that takes a bandpass pyramid `(detail,average)` produced by `i2p` and returns the original image at full resolution. Round the image at the end of the function `p2i` and cast it to `numpy.uint8`. For instance:

```
return numpy.round(img).astype(numpy.uint8)
```

Show your code and the reconstructed image, displayed with the provided function `show` called with the default keyword arguments.

## Answer

In [2]:
```
def p2i(detail, average):
    E_inv = np.array([[1, 1, 1, 1], [-1, 1, -1, 1],
                      [1, -1, -1, 1], [-1, -1, 1, 1]])
    for index in range(len(detail)-1, -1 ,-1):
        det =[]
        for i in range(3):
            det.append(np.array(detail[index][i]))
        det.append(average)
        Q = []
        for d in range(4):
            Q.append(np.zeros_like(det[0]))
            for q in range(4):
                Q[d] += E_inv[d, q] * det[q]
        A= (Q[0].shape[0]*2, Q[0].shape[0]*2)
        A= np.zeros(A)
        A[::2, ::2] = Q[0]
        A[1::2, ::2] = Q[1]
        A[::2, 1::2] = Q[2]
        A[1::2, 1::2] =Q[3]
        average = A
    img = average
    return np.round(img).astype(np.uint8)

image1 = p2i(detail,average)
show(image1)
```

# Part 4: Principal Component Analysis

## Problem 4.1

Use the function `scipy.linalg.eigh`, which computes eigenvalues and eigenvectors of a Hermitian matrix ("Hermitian" means symmetric if the matrix is real), to write a function with header

```
def pca(A, k):
```

that takes an $m \times n$ `numpy` array `A` of floats and an integer `k` and returns the triple `(mu, V, s)` defined as follows:

- `mu` is the mean of the rows of $A$
- `V` is the $n \times k$ matrix $V_k$ whose columns are the first $k$ right singular vectors of the centered version of $A$
- `s` is an $n$-dimensional vector with the standard deviations of $A$ along all principal axes, $s_i = \sigma_i/\sqrt{m-1}$ for $i = 1, \ldots, n$ (not just the first $k$).

Show your code and the result of the tests given in the template.

## Answer

In [25]:
```
%reset -f
import numpy as np
from scipy.linalg import eigh
# Your code here
def pca(A, k):
    [row,col] = A.shape
    mu = np.dot(np.ones([1,row]),A)/(row)
    C = A - np.dot(np.ones([row,1]),mu)
    S = np.dot(C.transpose(),C)
    [eival,eivec] = eigh(S)
    eival[eival < 0.0] = 0.0
    sigval = np.flip(eival**0.5)
    s = sigval/((row-1)**0.5)
    v = np.flip(eivec,1)[:,0:k]
    return mu,v,s
```

In [26]:
```
try:
    import numpy as np
    A = np.array([[1, 0, -2, 3],
                  [0, 2, 1, -1],
                  [-3, 1, 0, -1],
                  [0, 0, 1, 0],
                  [-1, 1, -1, 0]]).astype(np.float64)

    np.set_printoptions(precision=3, suppress=True)
    for k in range(1, 4):
        (mu, VT, s) = pca(A, k)
        print('k =', k, end='\n\n')
        print('mu =', mu, end='\n\n')
        print('VT =', VT, sep='\n', end='\n\n')
        print('s =', s, end='\n\n')
        print(30 * '=')
except NameError:
    pass
```

```
k = 1

mu = [[-0.6  0.8 -0.2  0.2]]

VT =
[[ 0.498]
 [-0.238]
 [-0.433]
 [ 0.713]]

s = [2.281 1.282 0.717 0.2  ]

==============================
k = 2

mu = [[-0.6  0.8 -0.2  0.2]]

VT =
[[ 0.498  0.774]
 [-0.238  0.1  ]
 [-0.433  0.611]
 [ 0.713 -0.135]]

s = [2.281 1.282 0.717 0.2  ]

==============================
k = 3

mu = [[-0.6  0.8 -0.2  0.2]]

VT =
[[ 0.498  0.774  0.214]
 [-0.238  0.1    0.862]
 [-0.433  0.611 -0.441]
 [ 0.713 -0.135 -0.13 ]]

s = [2.281 1.282 0.717 0.2  ]

==============================
```

## Problem 4.2

Write two functions with headers

```
def encode(A, mu, V):
```

and

```
def decode(B, mu, V):
```

that take the parameters `mu, V` of a PCA computed from some data set, as well as a matrix `A` of new, raw data (one data point per row) or a matrix `B` of PCA-encoded data and return the result of encoding `A` or decoding `B` through the given PCA. The new data points do *not* modify the existing PCA.

Show your code and the result of running the tests given in the template.

[Hint: Each function is a one-liner.]

## Answer

In [53]:
```
# Your code here
def encode(A,mu,V):
    return (np.dot(A - np.dot(np.ones([A.shape[0],1]),mu),V))

def decode(B, mu, V):
    return (np.dot(B,V.T)+np.dot(np.ones([A.shape[0],1]),mu))
```

In [55]:
```
try:
    A = np.array([[1, 0, -2, 3],
                  [0, 2, 1, -1],
                  [-3, 1, 0, -1],
                  [0, 0, 1, 0],
                  [-1, 1, -1, 0]]).astype(np.float64)

    mu, V, _ = pca(A, 4)
    np.set_printoptions(precision=2, suppress=True)
    for k in range(1, 5):
        VC = V[:, :k]
        AB = encode(A, mu, VC)
        AR = decode(AB, mu, VC)
        print('k =', k, end='\n\n')
        print('AB =', AB, sep='\n', end='\n\n')
        print('AR =', AR, sep='\n', end='\n\n')
        print(30 * '=')
except NameError:
    pass
```

```
k = 1

AB =
[[ 3.76]
 [-1.36]
 [-2.18]
 [-0.17]
 [-0.04]]

AR =
[[ 1.27 -0.09 -1.83  2.88]
 [-1.28  1.12  0.39 -0.77]
 [-1.69  1.32  0.75 -1.36]
 [-0.69  0.84 -0.12  0.08]
 [-0.62  0.81 -0.18  0.17]]

==============================
k = 2

AB =
[[ 3.76 -0.32]
 [-1.36  1.48]
 [-2.18 -1.55]
 [-0.17  1.14]
 [-0.04 -0.75]]

AR =
[[ 1.02 -0.13 -2.03  2.92]
```

```
 [-0.13  1.27  1.29 -0.97]
 [-2.89  1.16 -0.2  -1.15]
 [ 0.2   0.96  0.57 -0.08]
 [-1.2   0.74 -0.64  0.27]]

==============================
k = 3

AB =
[[ 3.76 -0.32  0.08]
 [-1.36  1.48  0.79]
 [-2.18 -1.55 -0.27]
 [-0.17  1.14 -1.06]
 [-0.04 -0.75  0.47]]

AR =
[[ 1.04 -0.06 -2.06  2.91]
 [ 0.04  1.95  0.95 -1.07]
 [-2.95  0.93 -0.08 -1.11]
 [-0.03  0.04  1.04  0.06]
 [-1.1   1.14 -0.85  0.21]]

==============================
k = 4

AB =
[[ 3.76 -0.32  0.08 -0.13]
 [-1.36  1.48  0.79 -0.11]
 [-2.18 -1.55 -0.27 -0.16]
 [-0.17  1.14 -1.06  0.09]
 [-0.04 -0.75  0.47  0.31]]

AR =
[[ 1. -0. -2.  3.]
 [-0.  2.  1. -1.]
 [-3.  1. -0. -1.]
 [ 0. -0.  1.  0.]
 [-1.  1. -1. -0.]]

==============================
```

## Problem 4.3

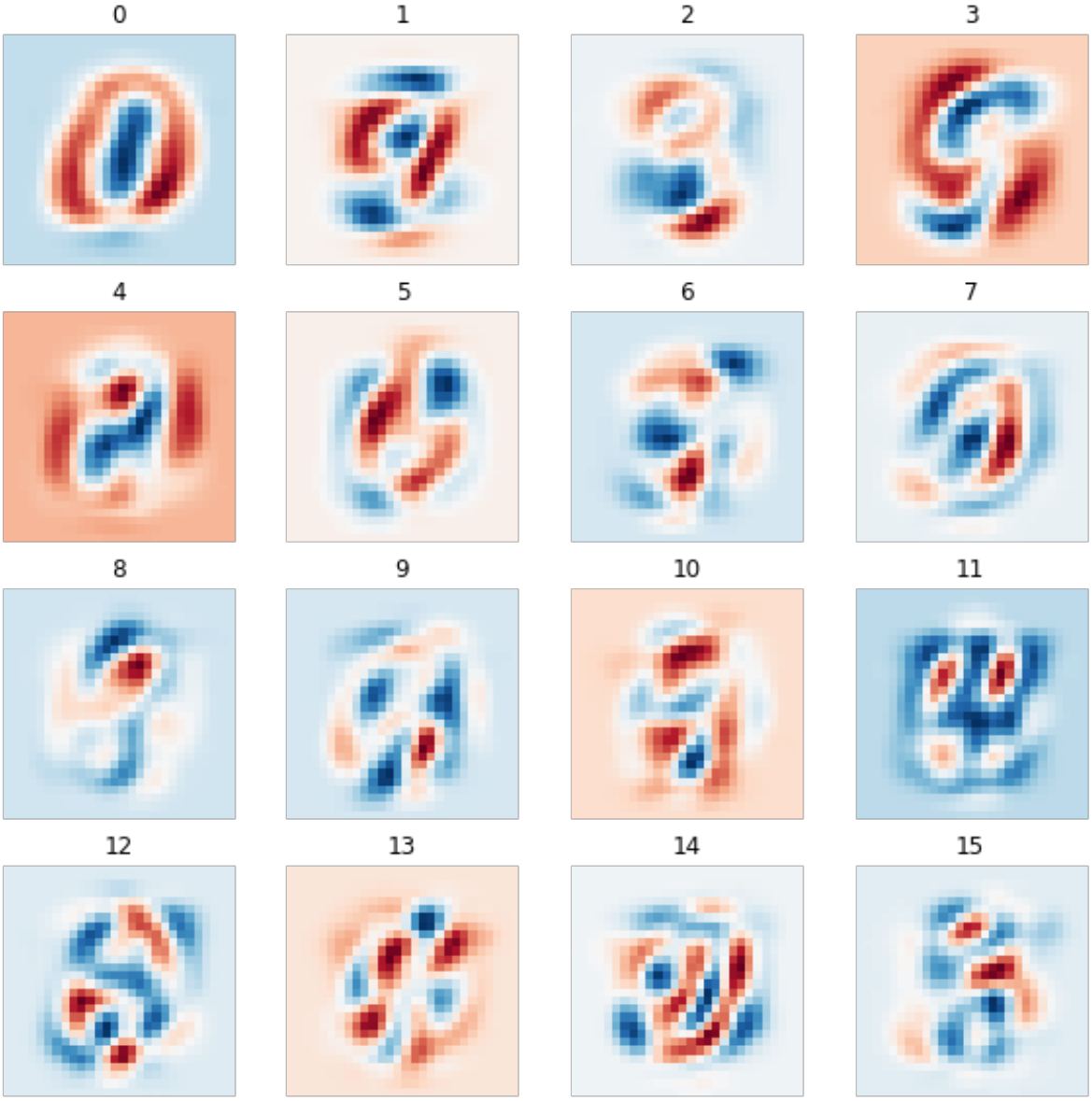
In [57]:
```
import pickle
import numpy as np

with open('digits.pickle', 'rb') as f:
    digits = pickle.load(f)

images, labels = digits['images'], digits['labels']

imgShape = images[0].shape
n = images[0].size
D = np.asarray([d.reshape((n,)) for d in images])
```

Use your function `pca` to compute the first 16 principal components (that is, a $784 \times 16$ matrix $V$) of the matrix `D`. Each column of $V$ has 784 numbers and can therefore be reshaped into a $28 \times 28$ image (of floating-point numbers with arbitrary signs).

Use `matplotlib.pyplot.subplot` to create a $4 \times 4$ array of plots that displays these 16 images. Use the `'RdBu'` color map in `matplotlib.pyplot.imshow`, turn off the axis labels with `matplotlib.pyplot.axis('off')`, and use `matplotlib.pyplot.title` to add a title to each plot that shows which component is being displayed (the title is just an integer between 0 and 15). A good figure size is obtained with `matplotlib.pyplot.figure(figsize=(10, 10))`.

## Answer

In [58]:
```
[mu,v,s] = pca(D,16)
from matplotlib import pyplot as plt
%matplotlib inline
plt.figure(figsize=(10,10))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(v[:,i].reshape((28,28)),cmap = 'RdBu')
    plt.axis('off')
    plt.title('%d'%i)
```



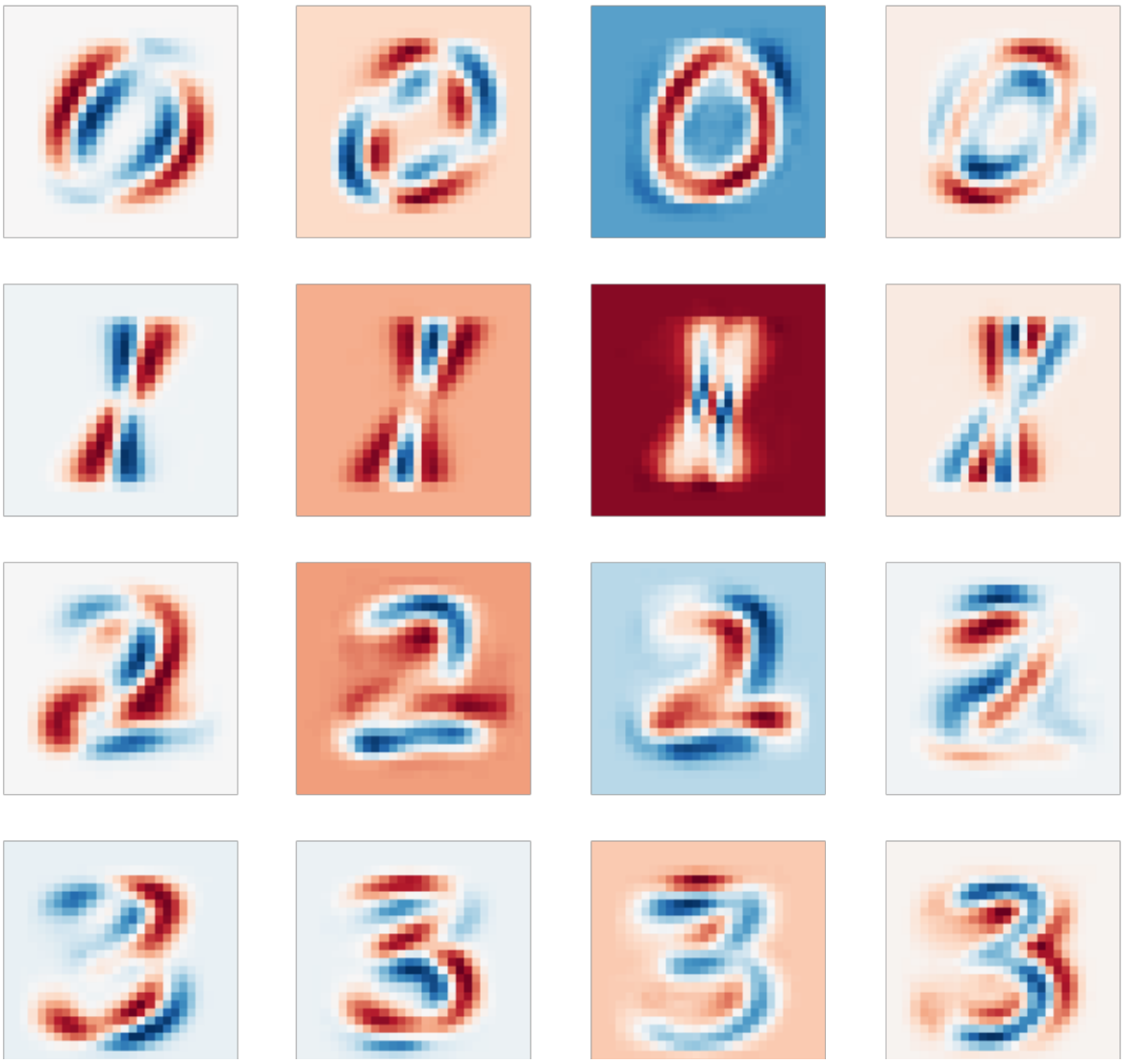

## Problem 4.4

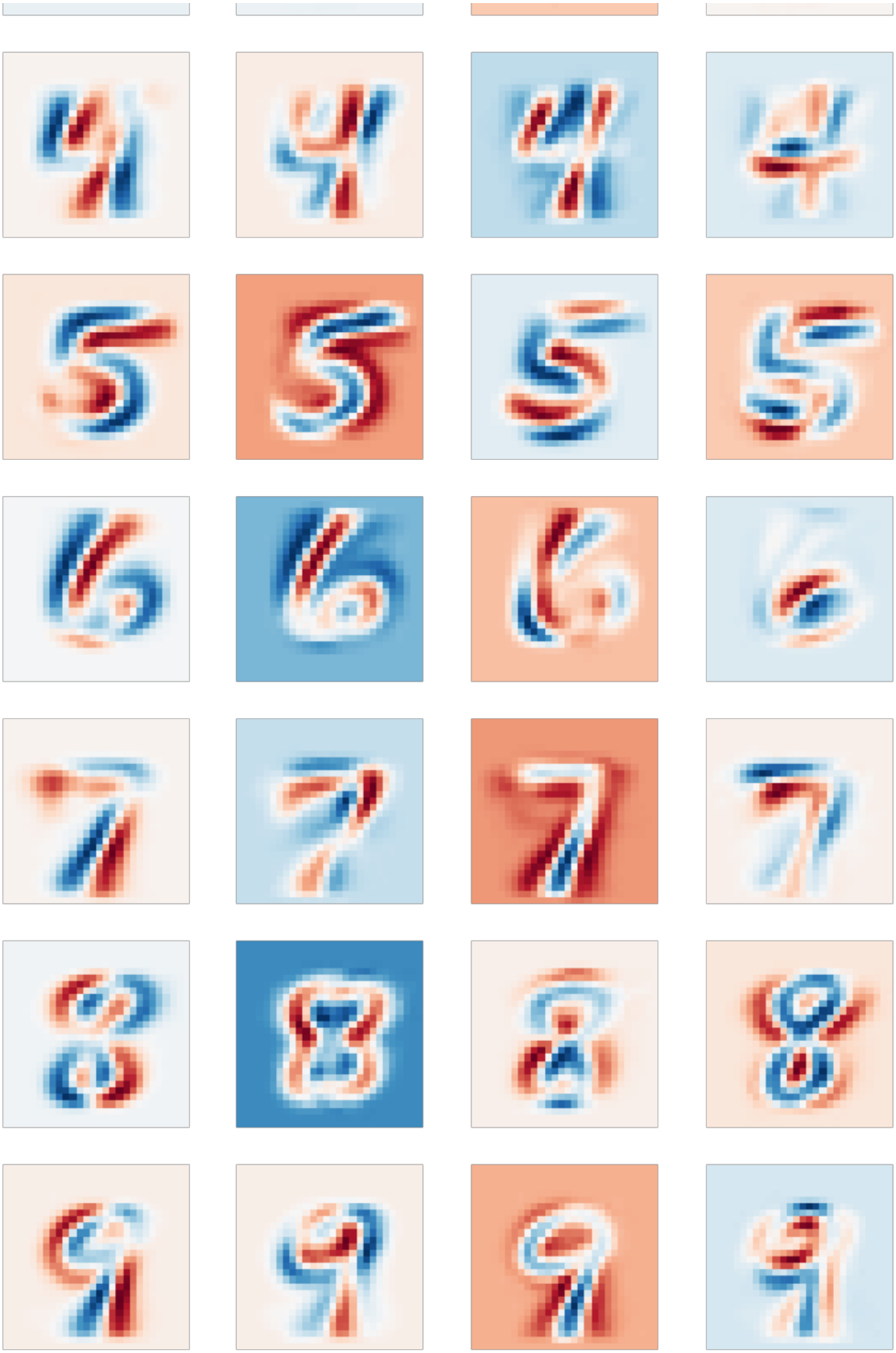
Make a 10 by 4 array of subplots (with `figsize=(10, 25)`) that for each digit from 0 to 9 shows the first four principal components for images of that digit only. Display these components as images, as you did in the previous problem.

Show code and resulting plots. No titles are needed.

## Answer

In [59]:
```
import pandas as pd
dataset = pd.DataFrame({'label': labels, 'images':list(D)},columns=['l
abel','images'])
plt.figure(figsize=(10,25))
j = 0
for i in range(10):
    result = dataset[dataset.label==i].images.values
    result = np.stack(result, axis = 1).T
    [mu,v,s] = pca(result,4)
    while (j < 4 * (i+1)):
        plt.subplot(10,4,j+1)
        plt.imshow(v[:,j%4].reshape((28,28)),cmap = 'RdBu')
        plt.axis('off')
        j+=1
```





## Problem 4.5

Encoding a matrix $A$ by PCA amounts to projecting $A - \boldsymbol{\mu}$ onto the columns of $V$, that is, to computing the inner product (or "template match score") of $A - \boldsymbol{\mu}$ and $V$. Write a couple of brief paragraphs explaining what the pictures you made in problem 4.4 tell you about the distibution of the shapes of the various digits, perhaps referring to an example from your display for problem 4.4.

## Answer

As mentioned in the problem statement, encoding matrix $A$ by PCA is same as computing the correlation between each centered data point and four unit norm vectors representing the four directions along where the data cloud has largest variance. Each column in $V$ represents an unit norm vector; Each entry in the column represents a dimension or pixel in the original image; and each image in each horizontal row represents one of the columns in the calculated $V$.

By looking at the images in problem 4.4, it can be easy to conclude that all principal components shows the shape of digits in the original images represented by a combination of positive and(or) negative values. This pattern is due to the fact that for the data set of each digit, the main differences between them occurs around the pixels that constitute the digits.

In addition, it is observed that the background all have the same color on the same plot with value equals to zero. This is due to the fact that the background pixels are not changed throughout the dataset and therefore has a zero variance. As a result, the principal components with largest variance does not incline along those axis and therefore have zero in that dimension(direction).

Comparing results between Problem 4.3 and Problem 4.4 exemplifies that PCA performs well when the data set does not occupies most locations in the whole space of original dimension as the dataset of every kind of digits in problem 4.4. Once the data occupies most of the space of original dimension, the value of the last few singular values are not trivial anymore, and the value of the first few singular values and relative right singular vectors are not representative of the whole dataset anymore as shown in the result of problem 4.4.