Deepayan Bhadra - hmwk4 Solutions

Q1(a): Optimality condition for L2 denoising

$\min \frac{\mu}{2}||\nabla x||^2 + \frac{1}{2}||x-b||^2$

Taking the gradient and setting it to zero gives the necessary condition:

$\mu\nabla^T\nabla(x) + x - b = 0$

Writing this in the form of a large linear system Ax = b gives

$(\mu\nabla^T\nabla + I)(x) = b$

Q1(b): Creating Function Handle for linear operator A

```
img= mpimg.imread('lena512.bmp')
img = img.astype(float)
b = img/max(img.flatten()) # Scaling to [0,1]
mu = 2
x0 = np.random.randn(*b.shape)
f = lambda x:mu*div2d(grad2d(x))+x-b
output = f(x0) # Evaluating A at x0
```

Q1(c): Richardson Iteration

```
def richardson(A,b,x,t):
    resids = 1
    all_res = []
    while resids > 10e-6:
        x = x+t*(b-A(x))
        resids = np.linalg.norm(b-A(x),'fro')
        all_res = np.append(all_res,resids)
    return all_res,x

t = 0.05
all_res,x = richardson(f,b,x0,t)
plt.xlabel('# of iterations')
plt.ylabel('Residual norm')
plt.plot(all_res)
print("# of iterations to convergence is",all_res.size)
plt.imshow(b) # Noisy image
plt.imshow(x.astype(float)) # De-noised image
```

No. of iterations to convergence is 334
(Plots at the end)

Q1(d): Conjugate Gradient

```
def conjgrad(A,b,x):
    tol = 10e-6;
    xk = x; rk = b-A(xk); pk = rk # Initial values
    res = [];
    while np.linalg.norm(rk)>tol:
        grad = A(pk);
        alpk = np.dot(rk.flatten(),rk.flatten())/(np.dot(pk.flatten(),
                                                grad.flatten()))
        xk = xk+alpk*pk;
        rkk = rk-alpk*grad;
        betak = np.dot(rkk.flatten(),rkk.flatten())/(np.dot(rk.flatten(),
                                                        rk.flatten()))
        pk = rkk+betak*pk;
        rk = rkk;
        res = np.append(res,np.linalg.norm(rk,'fro'))
        # Storing all the residuals

    return x,resids

[x,res] = conjgrad(f,b,x0)
plt.xlabel('# of iterations')
plt.ylabel('Residual norm')
plt.plot(all_res)
print("No of iterations to convergence is",all_res.size)
plt.imshow(b) # Noisy image
plt.imshow(x.astype(float)) # De-noised image
```

No. of iterations to convergence is 34
(Plots at the end)

Q1(e): With mu = 10

The iteration count increases almost two-fold. This is because, now significantly less weight (1/10) is allotted to the difference factor (x-b) (Images and convergence plots at the end)

Q1(f): Compute exact solutions using FFT

```
def l2denoise(b,mu):
    kernel = np.zeros((b.shape[0],b.shape[1]))
    kernel[0,0] = 1
```

```
    kernel[0,1] = -1
    Dx = np.fft.fft2(kernel)
    kernel = np.zeros((b.shape[0],b.shape[1]))
    kernel[0,0] = 1
    kernel[1,0] = -1
    Dy = np.fft.fft2(kernel)
    dd = np.divide(1,(mu*(np.conj(Dx)*Dx+np.conj(Dy)*Dy)+1))
    x = np.real(np.fft.ifft2(dd*np.fft.fft2(b)))
    return x

print('The norm of the gradient of the objective function is')
np.linalg.norm(f(l2denoise(b,mu)))
```

The norm of the gradient of the objective function is ans = 7.0224e-13

Q2(a): Building Two-Moons dataset

```
def make_moons(n):
    """Create a 'two moons' dataset with n feature vectors,
        and 2 features per vector."""

    assert n%2==0, 'n must be even'
    # create upper moon
    theta = np.linspace(-pi / 2, pi / 2, n//2)
    # create lower moon
    x = np.r_[np.sin(theta) - pi / 4, np.sin(theta)]
    y = np.r_[np.cos(theta), -np.cos(theta) + .5]
    data = np.c_[x, y]
    # Add some noise
    data = data + 0.03 * np.random.standard_normal(data.shape)

    # create labels
    labels = np.r_[np.ones((n//2, 1)), -np.ones((n//2, 1))]
    labels = labels.ravel().astype(np.int32)

    return data,labels

data,l = make_moons(100)
plt.scatter(data[l>0,0],data[l>0,1],c='r')
plt.scatter(data[l<0,0],data[l<0,1],c='b')
plt.show()
S = np.zeros((100,100))
sig = 0.09
B = pdist(data)
C = squareform(B)
S = np.exp(-C/sig)
```

Q2(b): Compute Normalized Similarity Matrix

```
S_sum = np.sum(S,axis=1,keepdims = True)
D = np.diagflat(S_sum)
temp = np.power(S_sum,-0.5)
S_hat = np.diagflat(temp).dot(S).dot(np.diagflat(temp))
```

Q2(c): Spectral Embedding

```
E,V = np.linalg.eig(S_hat)
idx = E.argsort()[::-1]
E = E[idx]
V = V[:,idx]

U2 = np.stack([V[:,1],V[:,2]],axis=1)
plt.scatter(U2[l==1,0],U2[l==1,1],c='r')
plt.scatter(U2[l==-1,0],U2[l==-1,1],c='b')
plt.show()
```

(Combined plots at the end)

Q2(d): k-means clustering

```
kmeans = KMeans(n_clusters = 2)
y_kmeans = kmeans.fit_predict(U2)
plt.scatter(U2[y_kmeans == 0, 0], U2[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(U2[y_kmeans == 1, 0], U2[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s = 300, c = 'yellow', label = 'Centroids')
```

Q3(a): Spectral Grouping Using Nystrom Method

```
data,l = make_moons(100000)
sig = 0.09
idx = np.random.choice(100000,size = 200, replace=False) # Random Sampling of 200 columns
idx_c = np.setdiff1d(np.arange(100000),idx)
temp1 = cdist(data,data[idx,:]) # Pairwise distance between two sets of observations
C = np.exp(-temp1/sig)
W = C[idx,:]
```

Q3(b): Forming Normalized C, W, M matrices

```
Z = C[200:,:]
W_m = np.sum(W,axis=1,keepdims = True)+ np.sum(np.transpose(Z),
                                                axis=1,keepdims = True)
M_e = np.sum(Z,axis=1,keepdims = True)+ np.transpose(np.sum(Z,
                                                axis=0,keepdims = True)
                                               .dot(np.linalg.inv(W))
                                               .dot(np.transpose(Z)))
diagD = np.concatenate([W_m,M_e],axis=0)
temp = np.multiply(np.repeat(diagD,200,axis=1),C)
C_n = np.dot(temp,np.diagflat(diagD[idx]))
W_n = C_n[idx,:]
M_n = C_n[idx_c,:]
```

Q3(c): Approximate eigenvectors

```
W_hat = W_n + sp.linalg.sqrtm(np.linalg.inv(W_n)).dot(np.transpose(M_n)).dot(M_n).
dot(sp.linalg.sqrtm(np.linalg.inv(W_n)))
# Orthogonalization matrix

D_w,U_w = np.linalg.eig(W_hat)
D_w,U_w = np.real(D_w), np.real(U_w)

# Eigen-decomposition W_hat = V*D*V'

U = np.matmul(C_n,np.matmul(sp.linalg.sqrtm(np.linalg.inv(W_n)),U_w))*
np.power(np.expand_dims(D_w,axis=-1),-0.5).T

idx = np.argsort(D_w)[::-1]
U_w = U_w[idx]
U = U[:,idx]


U = U/np.expand_dims(U[:,0],axis=-1)
U2 = np.expand_dims(U[:,1],axis=-1)
plt.scatter(U2[l==1],U2[l==1],c='r')
plt.scatter(U2[l==-1],U2[l==-1],c='b')
plt.show()
# Approximate eigen-vectors
```

(Combined plots at the end)

Q3(d): k-means

```
kmeans = KMeans(n_clusters = 2)
U2 = U[:,-3:-1]
```

```
y_kmeans = kmeans.fit_predict(U2)
plt.scatter(U2[y_kmeans == 0, 0], U2[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(U2[y_kmeans == 1, 0], U2[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow',
label = 'Centroids')
```

(a) Richardson: Noisy Image

(b) Richardson: De-noised Image

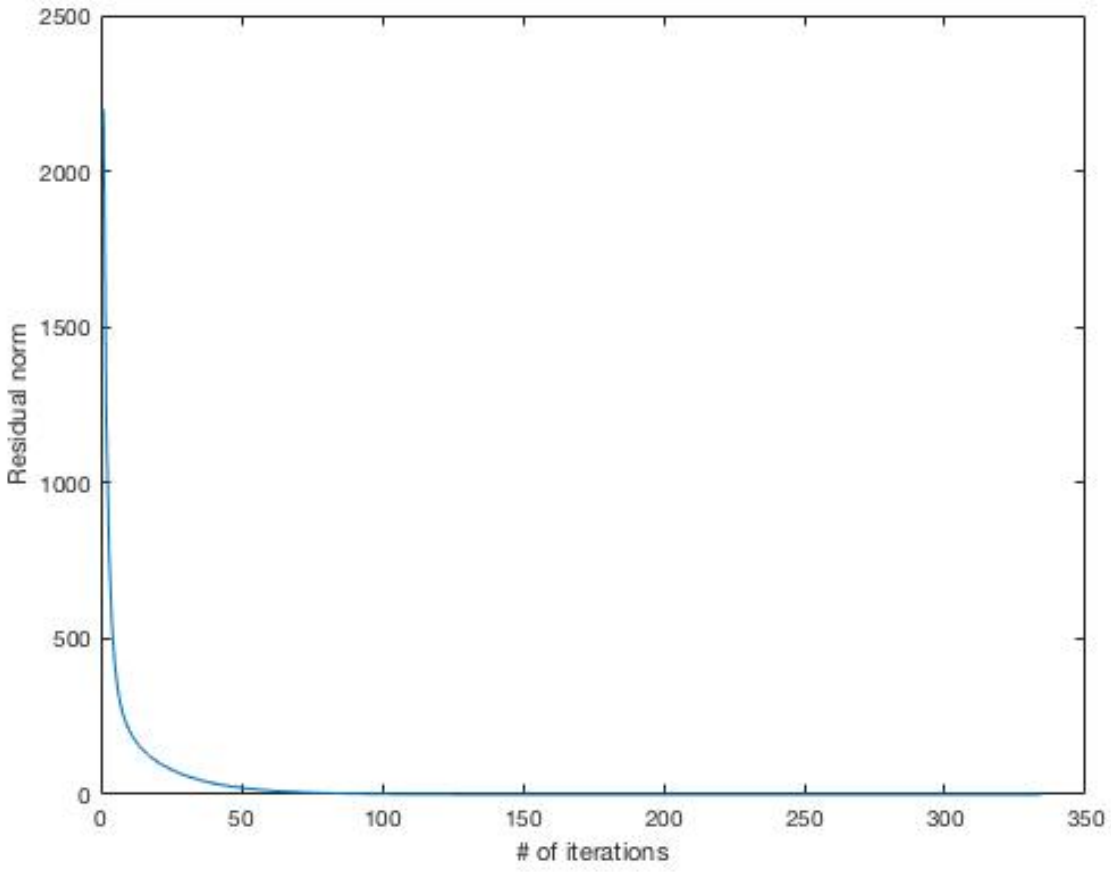


Figure 2: Richardson: Residual norm v/s No. of Iterations

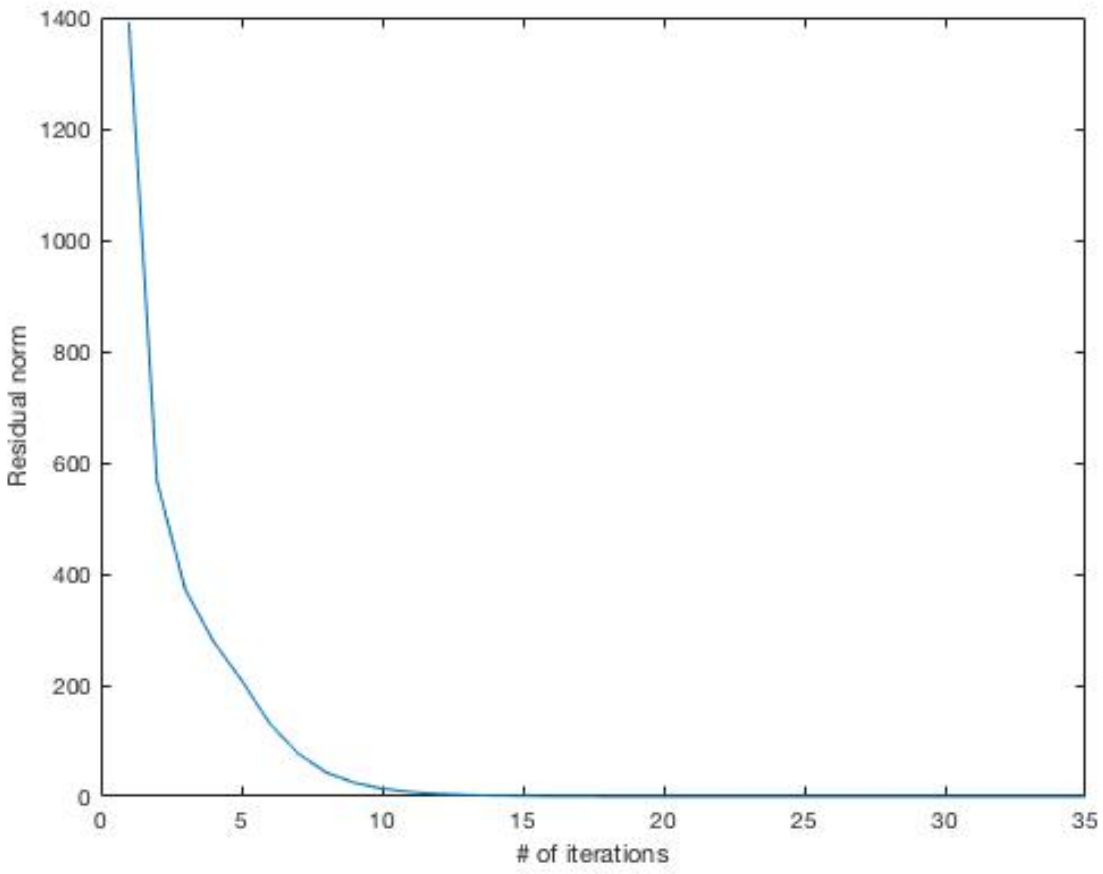


Figure 3: Conjugate Gradient: Residual norm v/s No. of Iterations

Figure 4: Conjugate Gradient: Noisy Image

Figure 5: Conjugate Gradient: De-noised Image

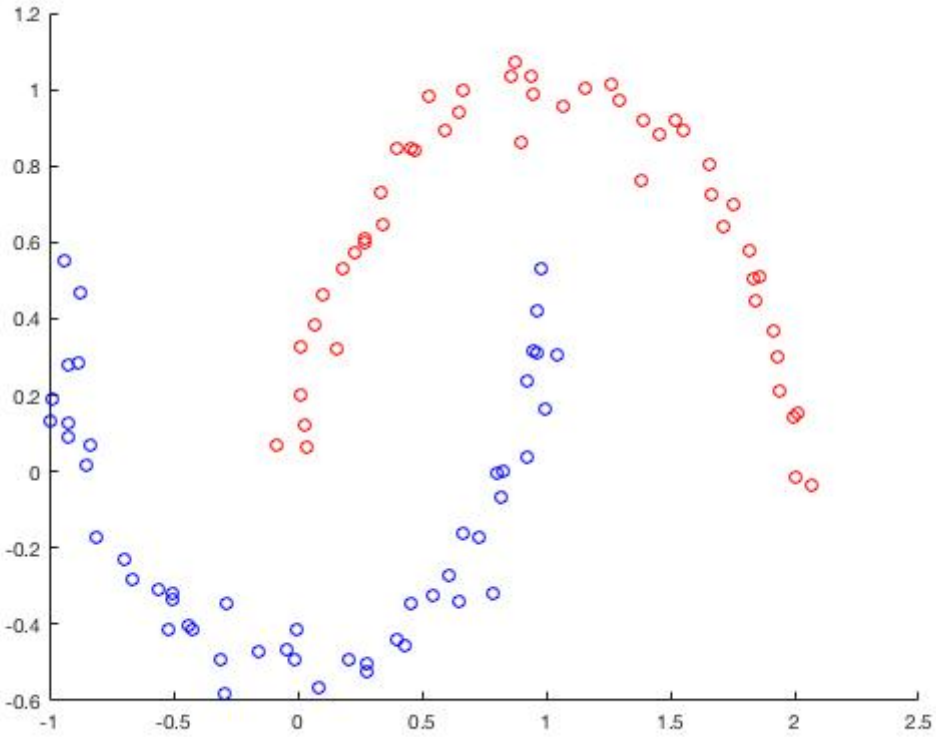
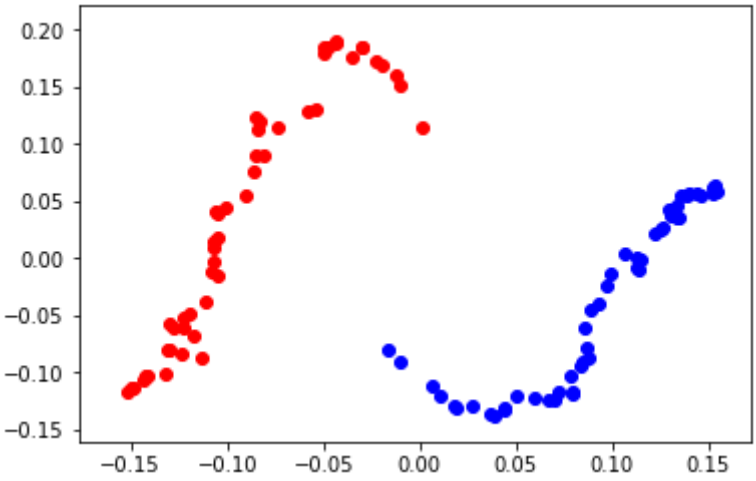
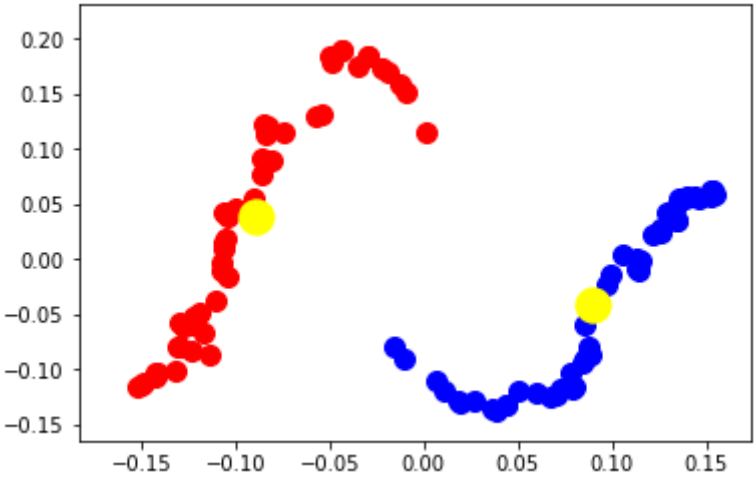


Figure 6: Small Dataset: Two Moons



(a) Spectral Embedding: Scattered columns of U2



(b) k-means clustering

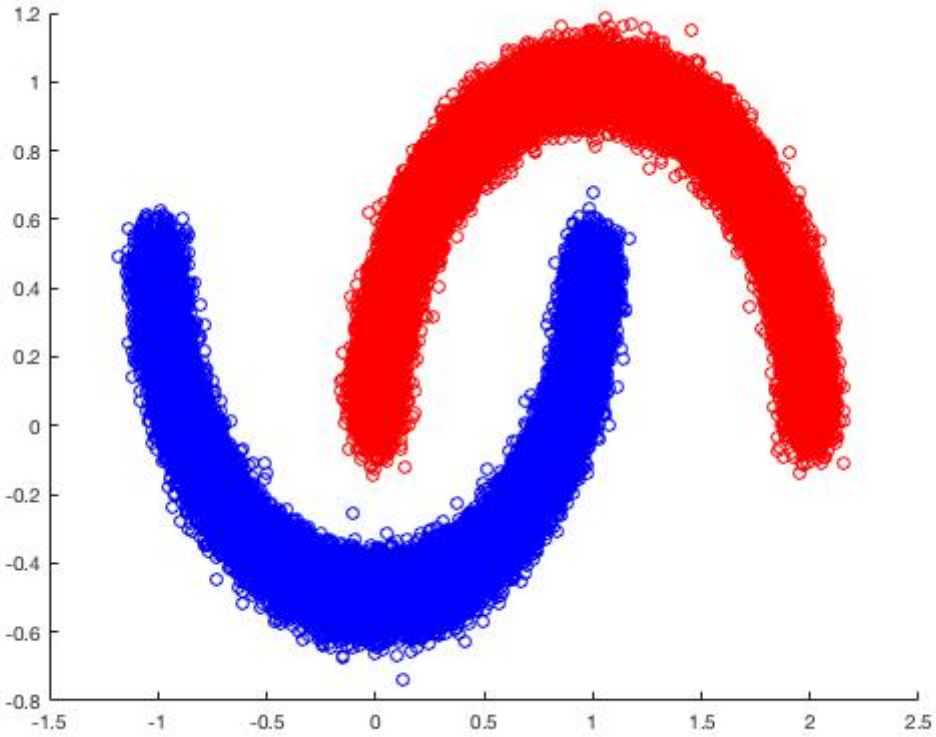
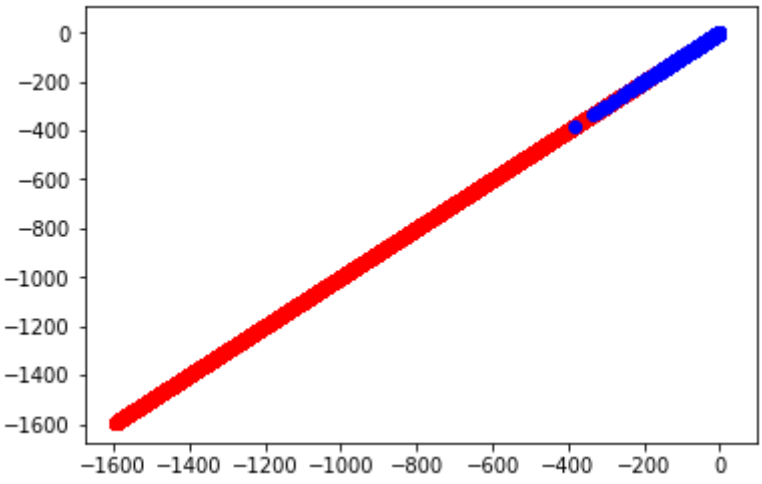
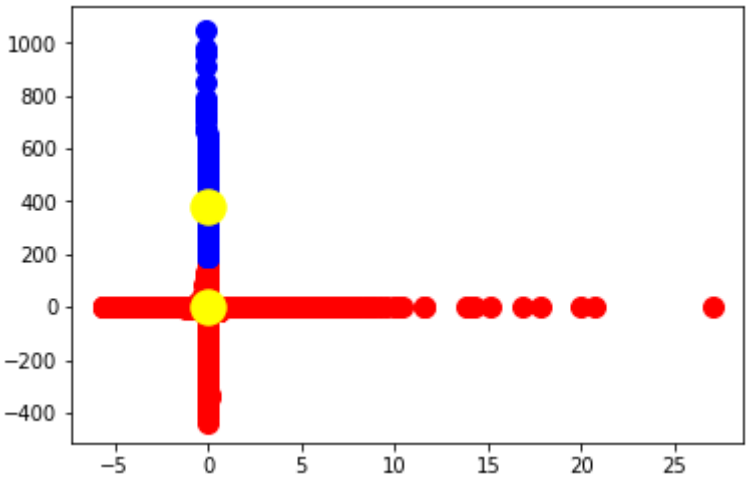


Figure 8: Large Dataset: Two Moons



(a) Approximate eigenvectors: Scattered columns of U2



(b) k-means clustering