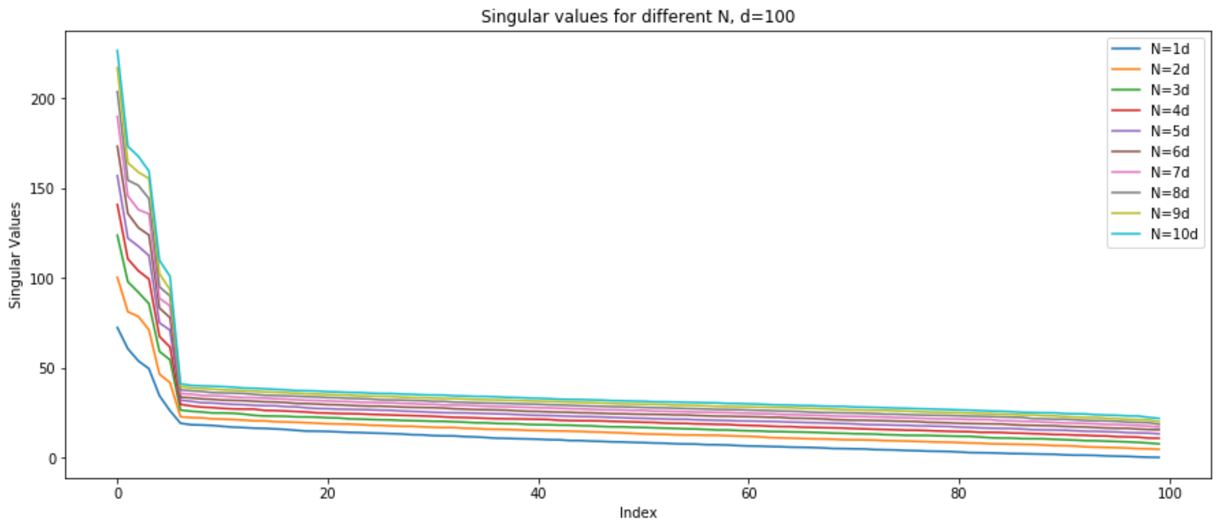
UNIVERSITY OF CALIFORNIA, SANTA BARBARA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Name: Ashish Vyas
Perm: 6931570
Course: ECE 283 Machine Learning
Homework: 4 PCA & Compressive Sensing

Part 1: PCA

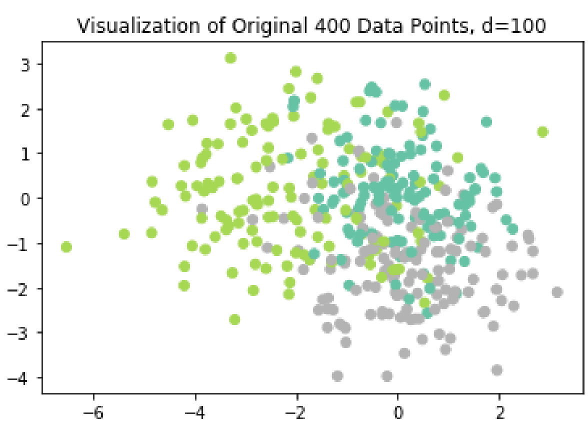
1.a. Generate N samples & perform SVD on *Nxd* matrix. How many dominant singular values do you see? How does this vary as you increase *N*, starting from say *N=2d* ?

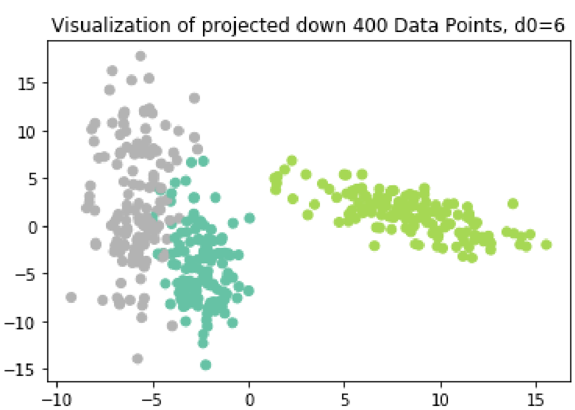


As we can see from plot above, increasing N does not change value of $d_0=6$.

1.b Perform PCA to obtain *Nxd₀* data matrix & implement Kmeans Algorithm

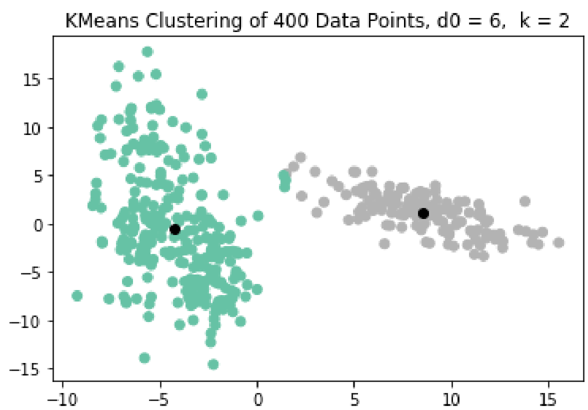
Plotting first 2 dimensions as x & y coordinates.



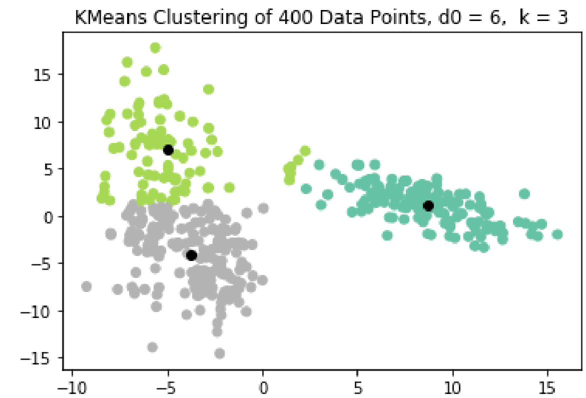


Here, Black point denotes cluster centroid.
Empirical probabilities are in form:

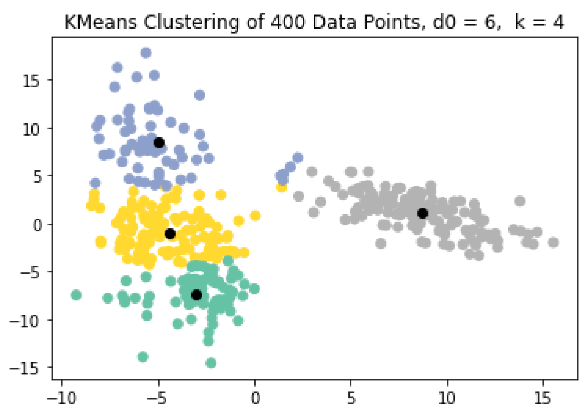
$$P[a_i[k] = 1 \mid z_i[l] = 1], where\ l = 1,2,3\ \&\ k = 1, \ldots, K\ in\ form\ of\ 3 \times K\ Table$$



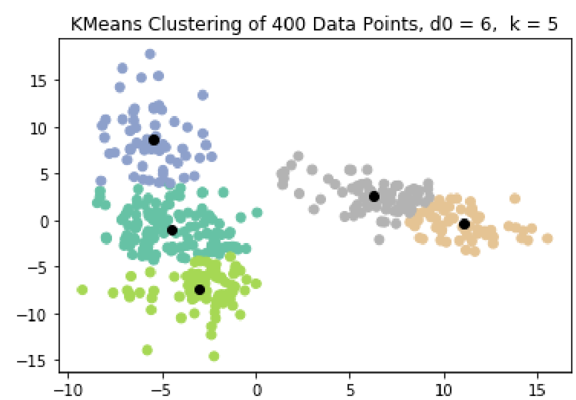

```
Empirical Probabilities :
[[0.3125 0.    ]
 [0.0075 0.33  ]
 [0.35   0.    ]]
```




```
Empirical Probabilities :
[[0.     0.025  0.2875]
 [0.3225 0.015  0.    ]
 [0.     0.1775 0.1725]]
```




```
Empirical Probabilities :
[[0.1625 0.01   0.14   0.    ]
 [0.     0.0125 0.0025 0.3225]
 [0.0325 0.1425 0.175  0.    ]]
```




```
Empirical Probabilities :
[[0.14   0.01   0.1625 0.     0.    ]
 [0.     0.     0.     0.15   0.1875]
 [0.175  0.1425 0.0325 0.     0.    ]]
```

Ground truth:
Comp 0 = 0.3125
Comp 1 = 0.3375
Comp 2 = 0.3500

2. Geometric Insight

For k = 2

All points from comp 0 are perfectly mapped onto 1st cluster.
Almost all points from comp 1 are mapped onto 2nd cluster.
All points from comp 2 are mapped onto 1st cluster.

Since comp 2 is a combination of comp 0 & comp 1 & since it has a strong influence of ($u_1 + u_2$), it is mapped to cluster 1 where comp 0 is mapped, which also has a strong influence of ($u_1 + u_2$).

For k = 3

Almost all points from comp 0 are mapped to 3rd cluster.
Almost all points from comp 1 are mapped to 1st cluster.
50% of points from comp 2 are mapped to 2nd cluster & 50% to 3rd cluster.

We can see comp 2 being divided into two clusters where comp 0 & comp 1 are mapped to.

For k = 4

About 50% of comp 0 is mapped to 1st cluster & remaining is mapped to 3rd cluster.
Almost all points from comp 1 is mapped to 4th cluster.
About 50% of comp 2 is mapped to 2nd cluster & remaining to 3rd.

Similarly, points from comp 0 & comp 1with strong influence of ($u_1 + u_2$) are mapped to 3rd cluster.

For k = 5

About 50% of comp 0 & 50% of comp 1 is mapped to 1st cluster.
About 50% of comp 0 has its own cluster (3)
45% of comp 1 has its own cluster (4)
55% of comp 1 has its own cluster (5)
About 40% of comp 2 has its own cluster (2)

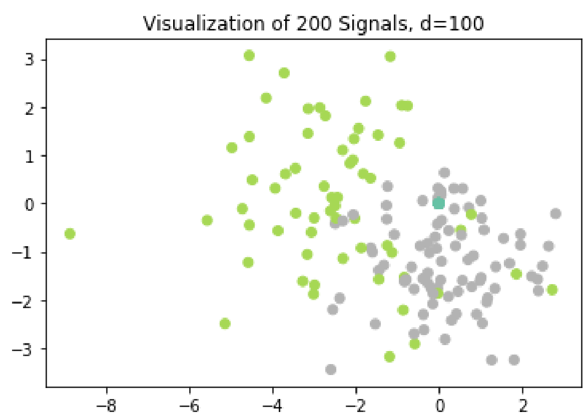
Part 2: Random Projections & Compressed Sensing.

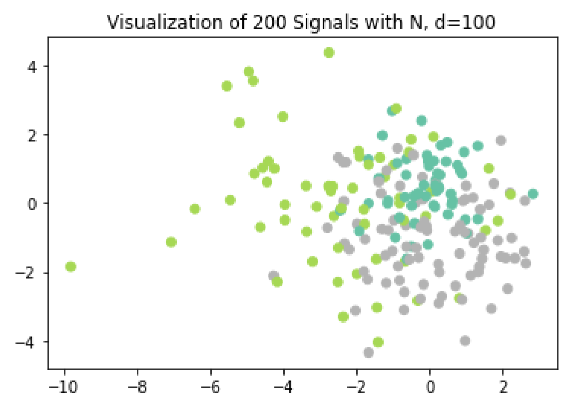
3. Generate *mxd* matrix Φ.

```
fi = np.random.choice([1,-1], (m,d), p=[0.5, 0.5])
```

a. Generate data & compute compressive projection.

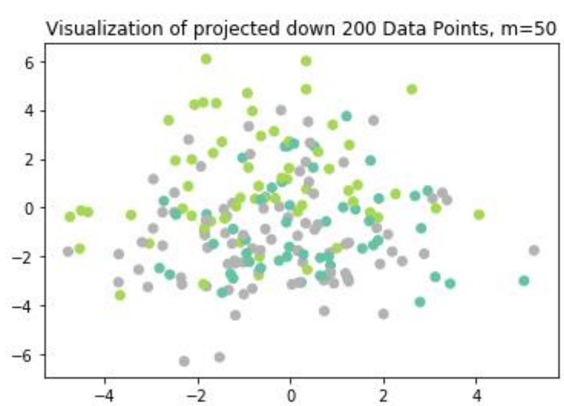
Generated N = 200 data points with dimension d = 100





Compressive Projection

$$\mathbf{y} = \frac{1}{\sqrt{m}}\Phi\mathbf{x}$$

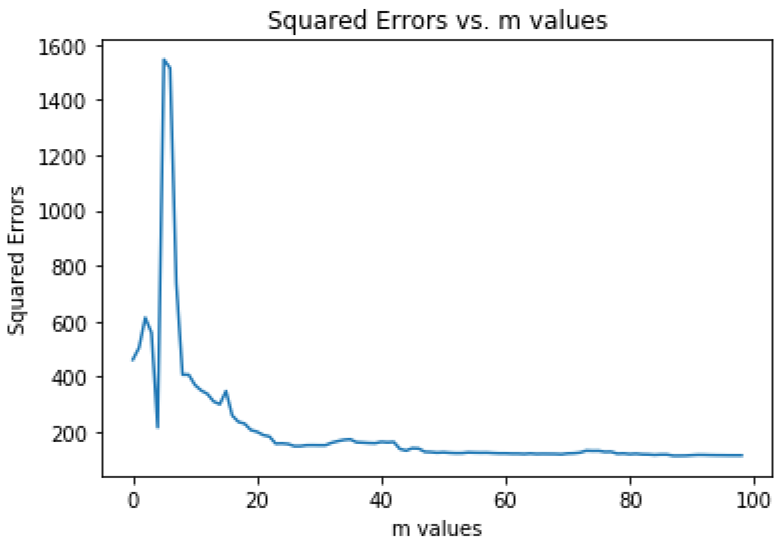


b. Define basis for the signal.

$$\mathbf{B} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4, \mathbf{u}_5, \mathbf{u}_6]$$

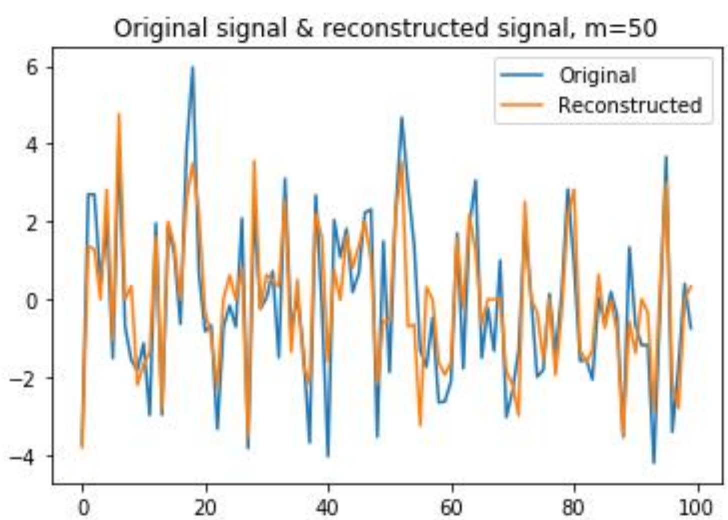
```
B = u.T
```

4. Find sparse reconstruction of s based on y. Play with the value of m until you get a satisfactory solution.



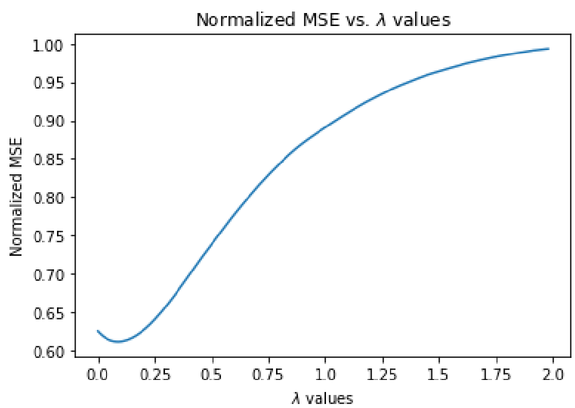

```
Best smallest m = 50
```

Calculated $\hat{s}$ based on different *m*, found smallest m with decent reconstruction = 50.

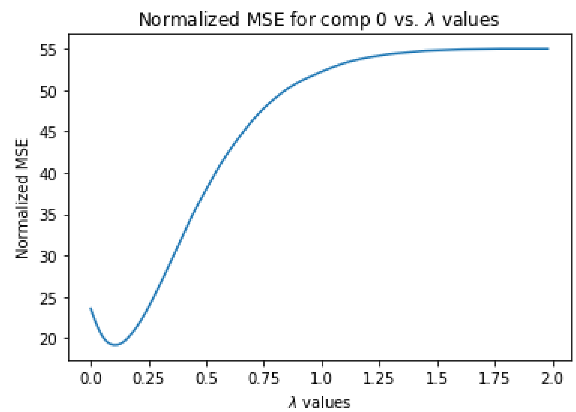


We can see, even when the signal is reconstructed from half the original dimensions, both signal matches closely.

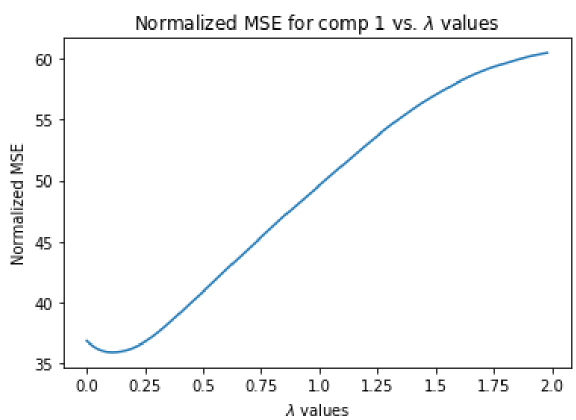
5. Compute the normalized MSE averaging over many draws. Plot normalized MSE vs. λ averaged over all data.



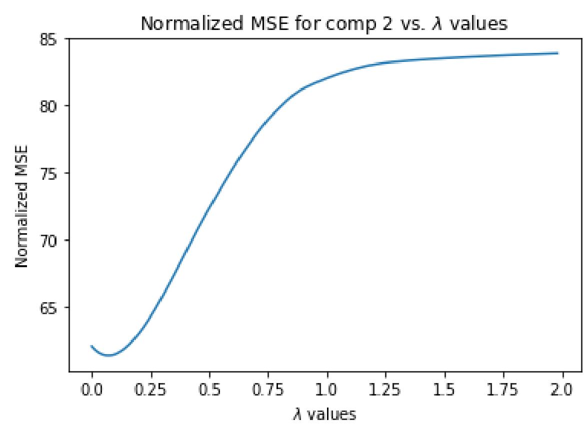

```
Best lambda = 0.09
```



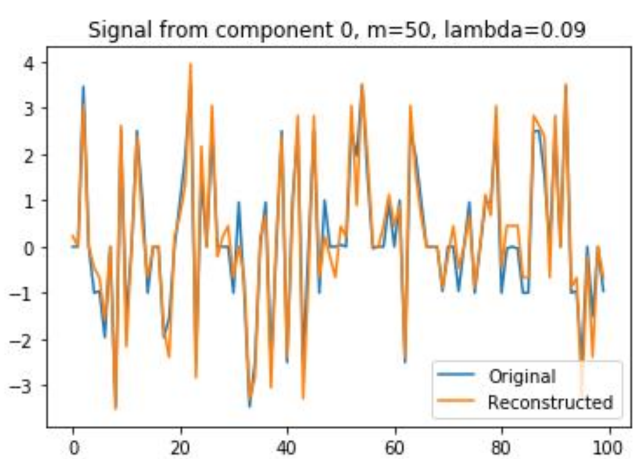

```
Best lambda for comp 0 = 0.1
```

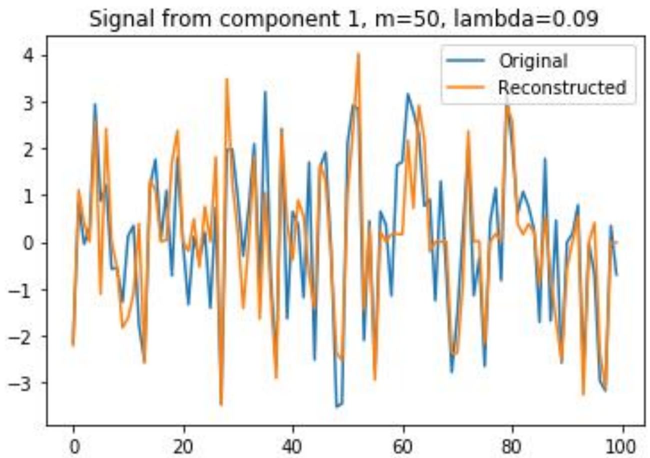


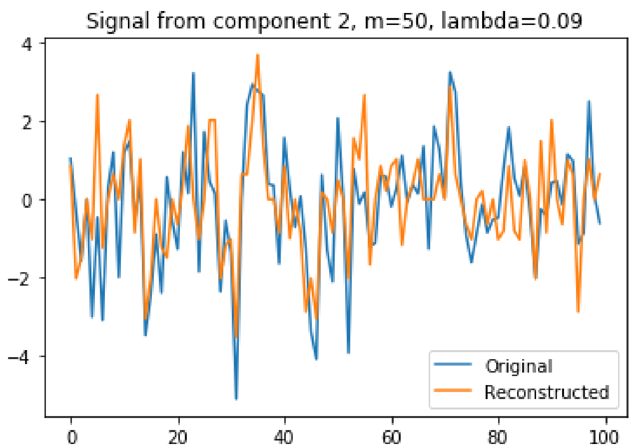

```
Best lambda for comp 1 = 0.11
```




```
Best lambda for comp 2 = 0.07
```

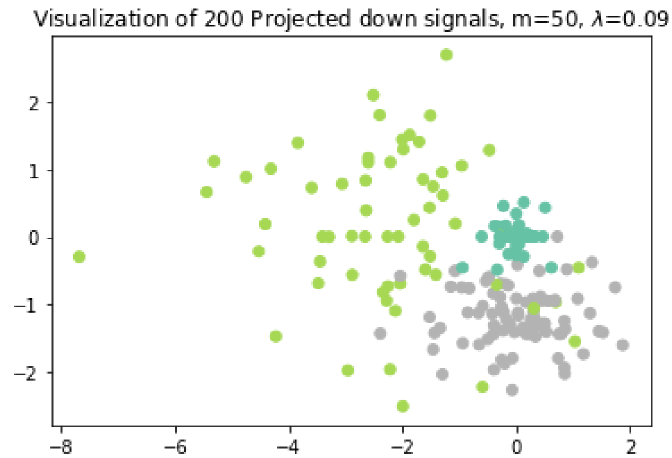


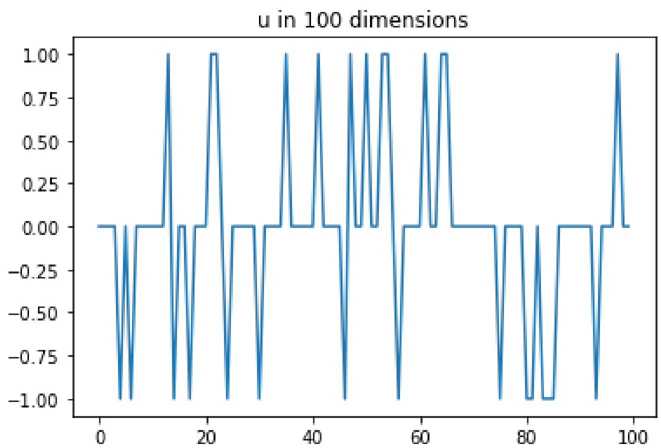


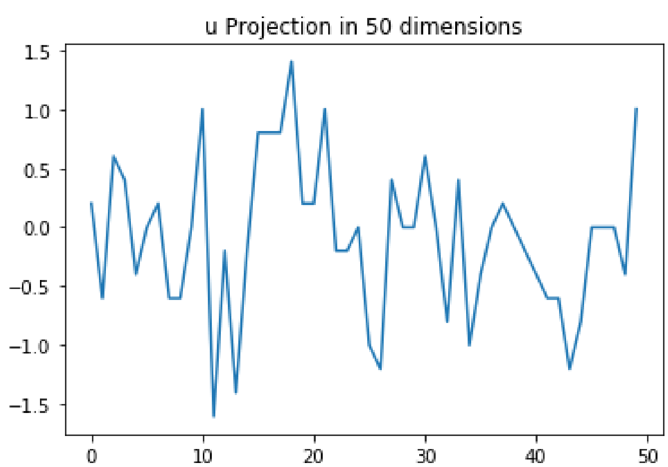


Comment: We can see that reconstruction is pretty good when we used *m* that was found in 4 along with lambda with minimum MSE over entire data.

6. For value of m found in 4, project data down to m dimensions. Compare the Euclidean distances squared vs. the corresponding quantities in the projected space.







```
Distance between projected u =
 [[0.         6.06530121 5.86205053 6.36911741 5.98482931 5.55050275]
 [6.06530121 0.         5.69954809 5.73664457 4.97671345 5.15222811]
 [5.86205053 5.69954809 0.         5.8895559  4.84507456 5.1912903 ]
 [6.36911741 5.73664457 5.8895559  0.         5.97131189 6.05863604]
 [5.98482931 4.97671345 4.84507456 5.97131189 0.         4.88245674]
 [5.55050275 5.15222811 5.1912903  6.05863604 4.88245674 0.        ]]

Distance between original u =
 [[0.         7.68114575 7.68114575 8.         7.54983444 7.54983444]
 [7.68114575 0.         7.87400787 8.18535277 7.74596669 7.74596669]
 [7.68114575 7.87400787 0.         8.18535277 7.74596669 7.74596669]
 [8.         8.18535277 8.18535277 0.         8.06225775 8.06225775]
 [7.54983444 7.74596669 7.74596669 8.06225775 0.         7.61577311]
 [7.54983444 7.74596669 7.74596669 8.06225775 7.61577311 0.        ]]
```

Element at (i,j) denotes Euclidean distance between u_i & u_j

Comment:

We can see that projected distances between two projected ui are kind of similar to that of original u_i.

Consider the ratio of 2 Euclidean distances at (1,4) & (1,5) of projected u:

$$\frac{6.3691}{5.9848} = 1.0642$$

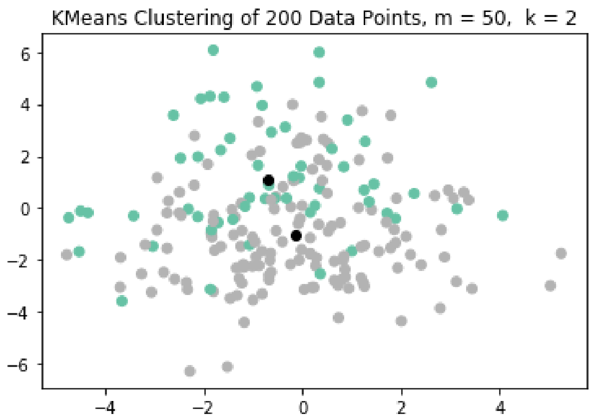
Consider the ratio of 2 Euclidean distances at (1,4) & (1,5) of original u:

$$\frac{8.0}{7.5498} = 1.0596$$

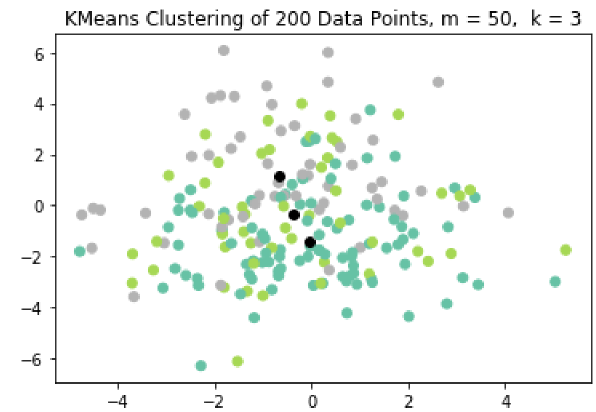
We can see that the ratios are almost equal. Therefore, we can say that overall shapes of the vectors are well preserved.

7. Implement the K-means algorithm with different values of K = 2,3,4,5 on projected data.

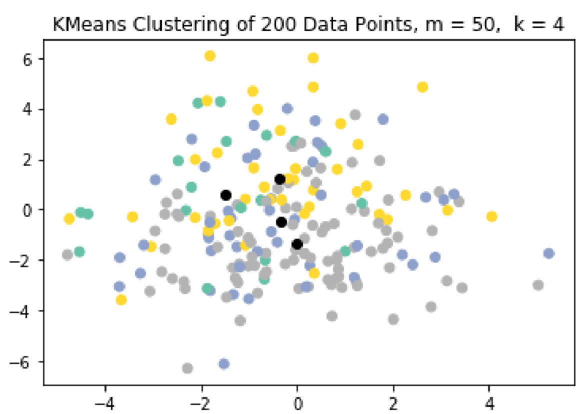
Ground truth:
Comp 0 = 0.275
Comp 1 = 0.305
Comp 2 = 0.420



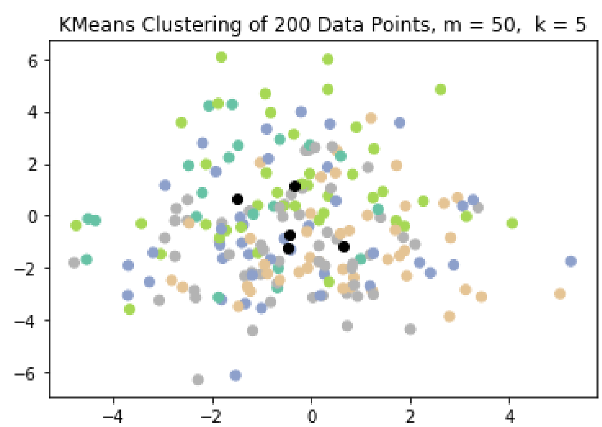

```
Empirical Probabilities :
[[0.    0.275]
 [0.285 0.02 ]
 [0.    0.42 ]]
```




```
Empirical Probabilities :
[[0.26  0.015 0.   ]
 [0.01  0.015 0.28 ]
 [0.2   0.22  0.   ]]
```




```
Empirical Probabilities :
[[0.    0.015 0.    0.26 ]
 [0.095 0.    0.21  0.   ]
 [0.    0.21  0.    0.21 ]]
```




```
Empirical Probabilities :
[[0.    0.    0.    0.2   0.075]
 [0.1   0.    0.205 0.    0.   ]
 [0.    0.2   0.    0.015 0.205]]
```

8. Geometric Insight on K-means

For k = 2

All points from comp 0 are perfectly mapped onto 2^{nd} cluster.
Almost all points from comp 1 are mapped onto 1^{st} cluster, some are mapped to cluster 2.
All points from comp 2 are mapped onto 2^{nd} cluster.

Since both comp 0 & comp 2 has a strong influence of ($u_1 + u_2$), entire comp 2 is mapped to cluster 2 where comp 0 is also mapped.

For k = 3

Almost all points from comp 0 are mapped to 1^{st} cluster.
Almost all points from comp 1 are mapped to 3^{rd} cluster.
Almost all points from comp 2 are mapped to 2^{nd} cluster.

For k = 4

Almost entire comp 0 and 50% of comp 2 is mapped to 4^{th} cluster.
70% of comp 1 has its own cluster (3)
30% of comp 1 has its own new cluster (1)
50% of comp 2 has its own cluster (2)

Intuitively, since there can be 4 clusters, points with strong ($u_1 + u_2$) are mapped to 4^{th} cluster.

For k = 5

About 72% of points from comp 0 are mapped to 4^{th} cluster.
Remaining 28% of comp 0 along with 50% of comp 2 is mapped to 5^{th} cluster.
About two-thirds of comp 1 has its own cluster (3), remaining one-third has its own cluster (1).
Remaining 50% of comp 2 has its own cluster (2).

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
```

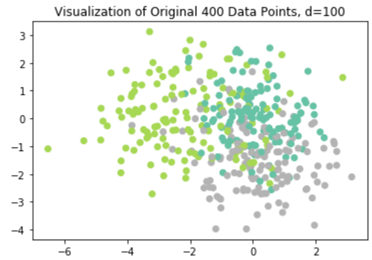
In [2]:
```python
def gen_d_data(data_pts, d=30, seed_no=30):

    np.random.seed(seed_no)

    #Generating Quasi-Orthogonal U
    u = np.zeros((6,d))
    for i in range(6):
        u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
        j = 0
        while (j<i):
            coeff = np.corrcoef(u[i],u[j])[0,1]
            if (coeff < -0.01 or coeff > 0.0):
                u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
                j = 0
            else:
                j += 1

    data = np.zeros((data_pts,d))
    labels = np.zeros((data_pts))

    for i in range(data_pts):
        c = np.random.randint(3, size=1)[0]
        if(c==0):
            data[i] = u[0] + np.random.normal(loc=0, scale=1)*u[1] +  np.random.normal(loc=0, scale=1)*u[2] +  np.random.normal(loc=0, scale=1, size=d)
            labels[i] = 0
        elif(c==1):
            data[i] = 2*u[3] + np.sqrt(2)*np.random.normal(loc=0, scale=1)*u[4] + np.random.normal(loc=0, scale=1)*u[5] + np.random.normal(loc=0, scale=1, size=d)
            labels[i] = 1
        else:
            data[i] = np.sqrt(2)*u[5] + np.random.normal(loc=0, scale=1)*(u[0]+u[1]) + (1/np.sqrt(2))*np.random.normal(loc=0, scale=1)*u[4] + np.random.normal(loc=0, scale=1,
size=d)
            labels[i] = 2

    return data,labels

def compute_KMeans(k,data):

    def update_labels(data, centroids):
        l = []
        for i in range(data.shape[0]):
            mse = []
            for j in range(centroids.shape[0]):
                mse.append(np.sqrt(np.sum(np.square(data[i] - centroids[j]))))
            l.append(np.argmin(mse))
        return l

    def new_centroids(data, centroids, new_labels):

        indices = []
        for j in range(centroids.shape[0]):
            indices.append([i for i, x in enumerate(new_labels) if x == j])

        new_centroids = []
        for i in range(len(indices)):
            new_centroids.append(np.sum(data[indices[i]], axis=0)/ len(indices[i]))

        return np.array(new_centroids)

    centroids = data[np.random.choice(data.shape[0], k, replace=False), :]

    old_labels = None
    new_labels = []
    while(new_labels != old_labels):
        old_labels = new_labels
        new_labels = update_labels(data, centroids)
        centroids = new_centroids(data, centroids, new_labels)

    return np.array(new_labels), centroids.T

def emp_prob(L, K, k):
    mat = np.zeros((3,k), dtype='float')
    for i in range(3):
        for j in range(k):
            s = 0
            for m in range(len(L)):
                if (L[m] == i and K[m] == j):
                    s = s+1
            mat[i,j] = s/float(len(L))
    return mat
```

In [48]:
```python
# d = 100
# p = 5
# sing_values = np.zeros((p,d))
# for i in range(1,p+1):
#     data, true_labels = gen_d_data(i*d, d)
#     u, sing_values[i-1], vh = np.linalg.svd(data)
# for i in range(p):
#     plt.plot(sing_values[i], label="N={}d".format(i+1))
# plt.title("Singular values for different N, d={}".format(d))
# plt.xlabel("Index")
# plt.ylabel("Singular Values")a
# plt.legend()
# plt.show()
```

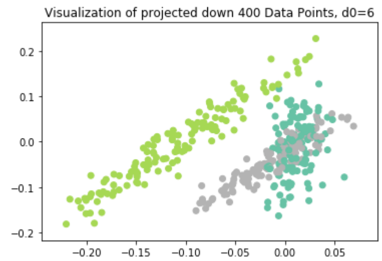
In [4]:
```python
N = 400
d = 100
data, true_labels = gen_d_data(N, d)
d0 = 6
```

In [5]:
```
def PCA(d0, data):

    u, s, vh = np.linalg.svd(data)
    a = np.matmul(u[:,:d0],s[:d0]*np.eye(d0))
    b = np.matmul(a,vh[:d0,:d0])
    c = np.matmul(b, vh[:d0,:d0].T)
    x_hat = np.matmul(c,vh.T[:d0,:d0])

    return x_hat

x_hat = PCA(d0, data)
```

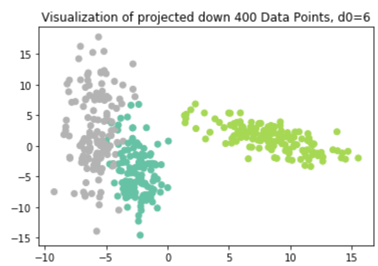
In [6]:
```
plt.scatter(data.T[0], data.T[1], c=true_labels, cmap=cm.Set2)
plt.title("Visualization of Original {} Data Points, d={}".format(N,d))
plt.show()
```



In [7]:
```
plt.scatter(x_hat.T[0], x_hat.T[1], c=true_labels, cmap=cm.Set2)
plt.title("Visualization of projected down {} Data Points, d0={}".format(N,d0))
plt.show()
```



In [43]:
```
k=2
pred_labels, centroids = compute_KMeans(k,x_hat)
plt.scatter(x_hat.T[0], x_hat.T[1], c=pred_labels, cmap=cm.Set2)
plt.plot(centroids[0], centroids[1], '.', markersize=12, color = 'black')
plt.title("KMeans Clustering of {} Data Points, d0 = {},  k = {}".format(N,d0,k))
plt.show()
print("Empirical Probabilities : \n{}".format(emp_prob(true_labels, pred_labels, k)))
```



In [1]:
```python
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from matplotlib import cm
```

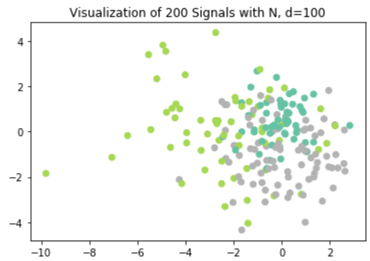
In [2]:
```python
def gen_d_data(data_pts, d=30):
    np.random.seed(seed_no)
    data = np.zeros((data_pts,d))
    data_with_noise = np.zeros((data_pts,d))
    labels = np.zeros((data_pts))

    for i in range(data_pts):
        c = np.random.randint(3, size=1)[0]
        if(c==0):
            data[i] = u[0] + np.random.normal(loc=0, scale=1)*u[1] +  np.random.normal(loc=0, scale=1)*u[2]
            data_with_noise[i] =  data[i] + np.random.normal(loc=0, scale=1, size=d)
            labels[i] = 0
        elif(c==1):
            data[i] = 2*u[3] + np.sqrt(2)*np.random.normal(loc=0, scale=1)*u[4] + np.random.normal(loc=0, scale=1)*u[5] + np.random.normal(loc=0, scale=1, size=d)
            data_with_noise[i] =  data[i] + np.random.normal(loc=0, scale=1, size=d)
            labels[i] = 1
        else:
            data[i] = np.sqrt(2)*u[5] + np.random.normal(loc=0, scale=1)*(u[0]+u[1]) + (1/np.sqrt(2))*np.random.normal(loc=0, scale=1)*u[4] + np.random.normal(loc=0, scale=1,
size=d)
            data_with_noise[i] =  data[i] + np.random.normal(loc=0, scale=1, size=d)
            labels[i] = 2

    return data, data_with_noise, labels

def compute_KMeans(k,data):

    def update_labels(data, centroids):
        l = []
        for i in range(data.shape[0]):
            mse = []
            for j in range(centroids.shape[0]):
                mse.append(np.sqrt(np.sum(np.square(data[i] - centroids[j]))))
            l.append(np.argmin(mse))
        return l

    def new_centroids(data, centroids, new_labels):

        indices = []
        for j in range(centroids.shape[0]):
            indices.append([i for i, x in enumerate(new_labels) if x == j])

        new_centroids = []
        for i in range(len(indices)):
            new_centroids.append(np.sum(data[indices[i]], axis=0)/ len(indices[i]))

        return np.array(new_centroids)

    centroids = data[np.random.choice(data.shape[0], k, replace=False), :]

    old_labels = None
    new_labels = []
    while(new_labels != old_labels):
        old_labels = new_labels
        new_labels = update_labels(data, centroids)
        centroids = new_centroids(data, centroids, new_labels)

    return np.array(new_labels), centroids.T

def emp_prob(L, K, k):
    mat = np.zeros((3,k), dtype='float')
    for i in range(3):
        for j in range(k):
            s = 0
            for m in range(len(L)):
                if (L[m] == i and K[m] == j):
                    s = s+1
            mat[i,j] = s/float(len(L))
    return mat
```

In [3]:
```python
d = 100
N = 200
seed_no = 30

np.random.seed(seed_no)

#Generating Quasi-Orthogonal U
u = np.zeros((6,d))
for i in range(6):
    u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
    j = 0
    while (j<i):
        coeff = np.corrcoef(u[i],u[j])[0,1]
        if (coeff < -0.01 or coeff > 0.0):
            u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
            j = 0
        else:
            j += 1

B = u.T

s, x, true_labels = gen_d_data(N, d)
```

In [158]:
```python
B.shape
```

Out[158]:
```
(100, 6)
```

In [4]:
```
plt.scatter(x.T[0], x.T[1], c=true_labels, cmap=cm.Set2)
plt.title("Visualization of {} Signals with N, d={}".format(N,d))
plt.show()
```

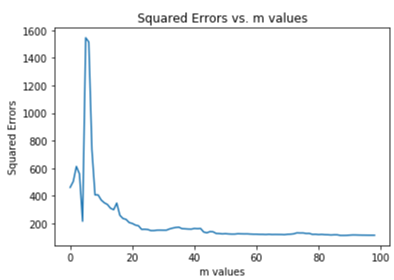


In [10]:
```
def comp_projection(m):    # gives compressed x !
    np.random.seed(seed_no)
    fi = np.random.choice([1,-1], (m,d), p=[0.5, 0.5])
    y = np.zeros((N,m))
    for i in range(N):
        y[i] = (1/np.sqrt(m)) * np.matmul(fi,x[i])
    return y, fi

def get_s_hat(m, alpha): # gives reconstruction of s
    y, fi = comp_projection(m)
    clf = linear_model.Lasso(alpha)
    clf.fit(((1/np.sqrt(m)) * np.matmul(fi,B)), y.T)
    s_hat = (np.matmul(B,clf.coef_.T)).T
    return s_hat
```

In [11]:
```
list_of_m_errors = []
for m in range(1,100):
    s_hat = get_s_hat(m, 0.1)
    list_of_m_errors.append(np.sum(np.square(s[0] - s_hat[0])))
```

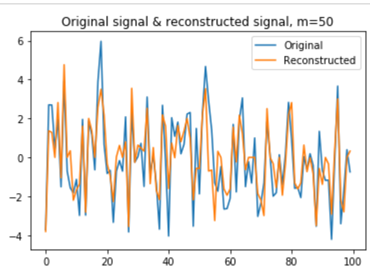
```
/data/home/ashishvyas/env1/local/lib/python2.7/site-packages/sklearn/linear_model/coordinate_descent.py:492: ConvergenceWarning: Objective did not converge. You might want t
o increase the number of iterations. Fitting data with very small alpha may cause precision problems.
  ConvergenceWarning)
```

In [12]:
```
plt.plot(list_of_m_errors)
plt.title("Squared Errors vs. m values")
plt.xlabel("m values")
plt.ylabel("Squared Errors")
plt.show()

best_m = np.argmin(list_of_m_errors[40:50]) + 41
print("Best smallest m = {}".format(best_m))
```




```
Best smallest m = 50
```

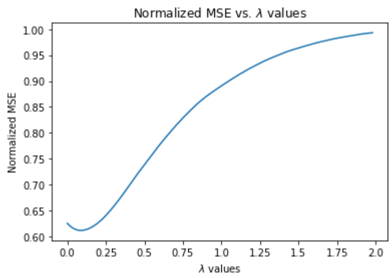
In [81]:
```
plt.plot(s[0], label='Original')
plt.plot(get_s_hat(best_m, 0.1)[0], label='Reconstructed')
plt.title("Original signal & reconstructed signal, m={}".format(best_m))
plt.legend()
plt.show()
```



In [69]:
```
print(true_labels[:20])
```

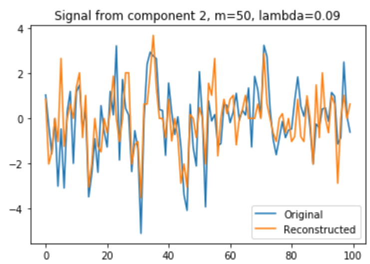
```
[1. 1. 2. 2. 2. 1. 2. 2. 2. 0. 2. 1. 2. 1. 2. 0. 1. 2. 1. 2.]
```

In [14]:
```
mse = []
for i in range(1,200):
    s_hat = get_s_hat(best_m, i/100.0)
    mse.append(np.linalg.norm(s - s_hat) / np.linalg.norm(s))

plt.plot(mse)
plt.title("Normalized MSE vs. $\lambda$ values")
plt.xlabel("$\lambda$ values")
plt.ylabel("Normalized MSE")
plt.xticks(np.arange(0,201,25),np.arange(0,2.1, 0.25))
plt.show()

best_lambda = np.argmin(mse)/100.0
print("Best lambda = {}".format(best_lambda))
```

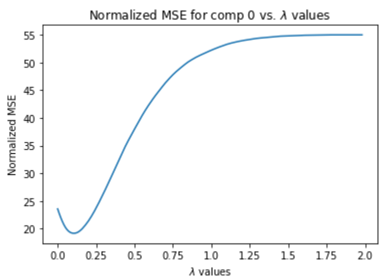


Best lambda = 0.09

In [101]:
```
plt.plot(s[12], label='Original')
plt.plot(get_s_hat(best_m, best_lambda)[12], label='Reconstructed')
plt.title("Signal from component {}, m={}, lambda={}".format(int(true_labels[12]),best_m, best_lambda))
plt.legend()
plt.show()
```

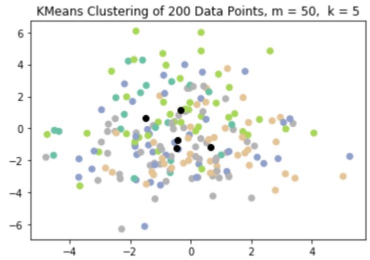


In [154]:
```
comp = 0
indices = np.where(true_labels == comp)[0]

mse = []
for i in range(1,200):
    s_hat = get_s_hat(best_m, i/100.0)
    msei = []
    for j in range(indices.shape[0]):
        msei.append(np.linalg.norm(s[indices[j]] - s_hat[indices[j]]) / np.linalg.norm(s[indices[j]]))
    mse.append(np.sum(np.array(msei)))

plt.plot(mse)
plt.title("Normalized MSE for comp {} vs. $\lambda$ values".format(comp))
plt.xlabel("$\lambda$ values")
plt.ylabel("Normalized MSE")
plt.xticks(np.arange(0,201,25),np.arange(0,2.1, 0.25))
plt.show()

best_lambda = np.argmin(mse)/100.0
print("Best lambda for comp {} = {}".format(comp, best_lambda))
```



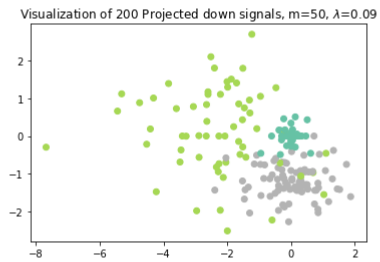
Best lambda for comp 0 = 0.1

```
In [149]: #7
k=5
y, _ = comp_projection(best_m)
pred_labels, centroids = compute_KMeans(k,y)
plt.scatter(y.T[0], y.T[1], c=pred_labels, cmap=cm.Set2)
plt.plot(centroids[0], centroids[1], '.', markersize=12, color = 'black')
plt.title("KMeans Clustering of {} Data Points, m = {},  k = {}".format(y.shape[0], best_m, k))
plt.show()
print("Empirical Probabilities : \n{}".format(emp_prob(true_labels, pred_labels, k)))
```




```
Empirical Probabilities :
[[0.    0.    0.    0.2   0.075]
 [0.1   0.    0.205 0.    0.   ]
 [0.    0.2   0.    0.015 0.205]]
```

```
In [16]: s_hat = get_s_hat(best_m, best_lambda)
plt.scatter(s_hat.T[0], s_hat.T[1], c=true_labels, cmap=cm.Set2)
plt.title("Visualization of {} Projected down signals, m={}, $\lambda$={}".format(N,best_m,best_lambda))
plt.show()
```



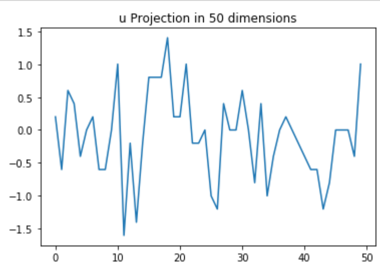

```
In [31]: # Projected down u
np.random.seed(seed_no)
fi = np.random.choice([1,-1], (best_m,d), p=[0.5, 0.5])
u_proj = (1/np.sqrt(m)) * np.matmul(fi,u.T)
print(u.shape)
```

```
(6, 100)
```

```
In [60]: euc_proj = np.zeros((6,6))
for i in range(6):
    for j in range(6):
        euc_proj[i,j] = np.linalg.norm(u_proj.T[i] - u_proj.T[j])
print("Distance between projected u =\n {}\n".format(euc_proj))

euc = np.zeros((6,6))
for i in range(6):
    for j in range(6):
        euc[i,j] = np.linalg.norm(u[i] - u[j])
print("Distance between original u = \n {}".format(euc))
```

```
Distance between projected u =
 [[0.         6.06530121 5.86205053 6.36911741 5.98482931 5.55050275]
 [6.06530121 0.         5.69954809 5.73664457 4.97671345 5.15222811]
 [5.86205053 5.69954809 0.         5.8895559  4.84507456 5.1912903 ]
 [6.36911741 5.73664457 5.8895559  0.         5.97131189 6.05863604]
 [5.98482931 4.97671345 4.84507456 5.97131189 0.         4.88245674]
 [5.55050275 5.15222811 5.1912903  6.05863604 4.88245674 0.        ]]

Distance between original u =
 [[0.         7.68114575 7.68114575 8.         7.54983444 7.54983444]
 [7.68114575 0.         7.87400787 8.18535277 7.74596669 7.74596669]
 [7.68114575 7.87400787 0.         8.18535277 7.74596669 7.74596669]
 [8.         8.18535277 8.18535277 0.         8.06225775 8.06225775]
 [7.54983444 7.74596669 7.74596669 8.06225775 0.         7.61577311]
 [7.54983444 7.74596669 7.74596669 8.06225775 7.61577311 0.        ]]
```

```
In [68]: plt.plot(u_proj.T[0])
plt.title("u Projection in {} dimensions".format(best_m))
plt.show()
```



In [67]:
```
plt.plot(u[0])
plt.title("u in {} dimensions".format(d))
plt.show()
```