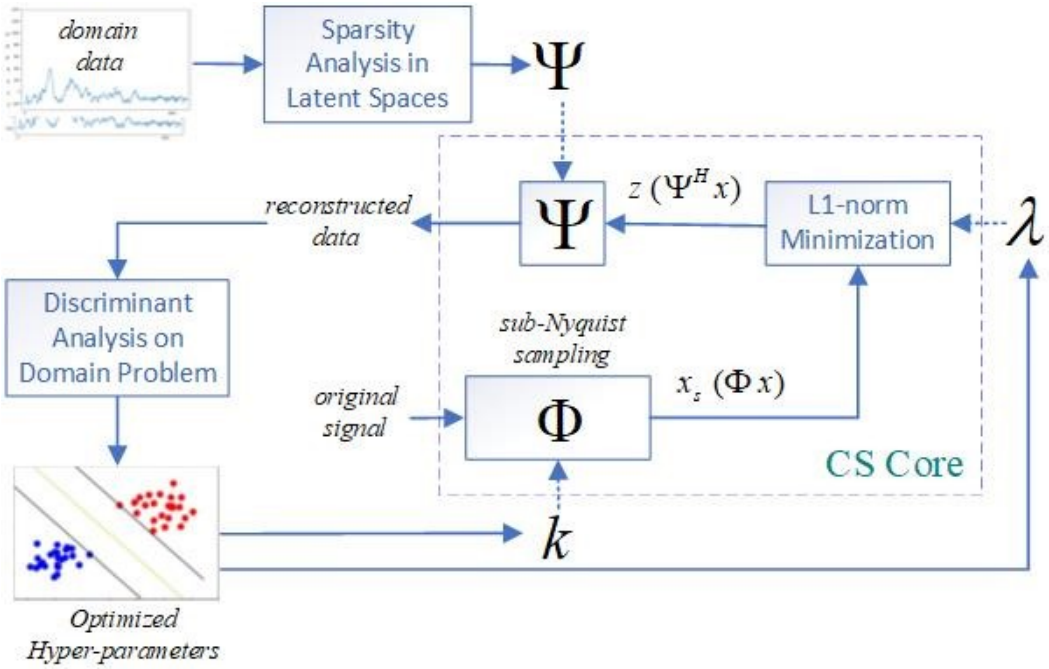
Adaptive Compressed Sensing of Raman Spectroscopic Profiling Data for Discriminative Tasks



Core Compressed Sensing (CS) Process

Sampling
$x_s = \Phi x$

Sparsity hypothesis
$x = \Psi z$

Compressed Sensing / sub-Nyquist Sampling
$x_s = \Phi \Psi z$

Reconstruction of latent z
$minimize \quad ||z||_1 \quad s.t. \quad x_s = \Phi \Psi z$

Inverse-transform to orginal space
$x = \Psi z$

Load Data

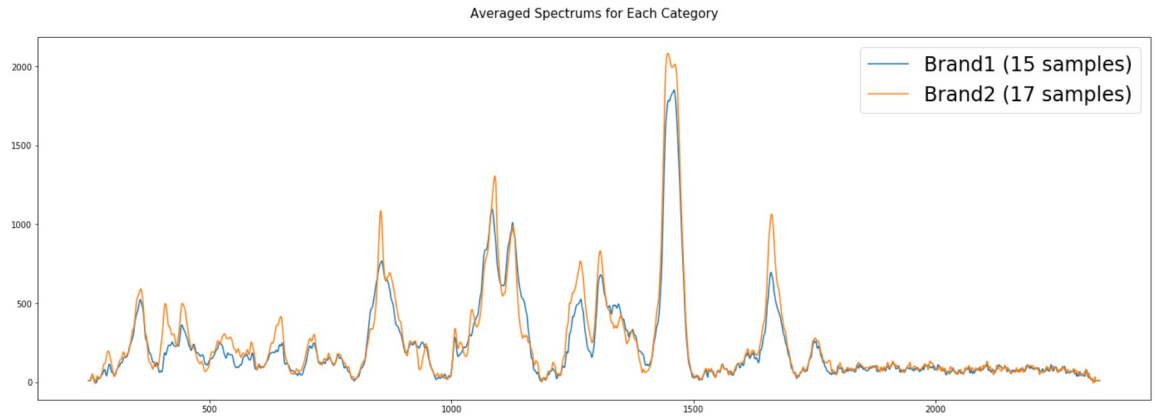
In [5]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.linear_model import Lasso
%matplotlib inline

path = os.getcwd() + "/7341_C2.txt"
data = pd.read_csv(path) # ,header=None
print(data.head())

cols = data.shape[1]
# convert from pandas dataframe to numpy matrices
X = np.matrix(data.iloc[:,1:cols].values)
y = np.array(data.iloc[:,0].values.ravel()) # first col is y label

# use map(float, ) to convert the string list to float list
X_names = list(map(float, data.columns.values[1:])) # X_names = np.array(list(da
ta)[1:])
labels = list(set(y))

m = X.shape[0]
n = X.shape[1]
```

```
   Label  250  251  252  253  254         255         256         257  \
0      1   10   10   10   10   10   43.570310   31.487300   16.338870
1      1   10   10   10   10   10    6.282227   -4.985352   -6.870117
2      1   10   10   10   10   10  129.908200  123.578100   92.240230
3      1   10   10   10   10   10  105.136700   93.134770   60.101560
4      1   10   10   10   10   10   66.738280   68.101560   64.845700

          258  ...  2330  2331  2332  2333  2334  2335  2336  2337  2338  2339
0  10.359380  ...    10    10    10    10    10    10    10    10    10    10
1   2.167969  ...    10    10    10    10    10    10    10    10    10    10
2  50.460940  ...    10    10    10    10    10    10    10    10    10    10
3  18.257810  ...    10    10    10    10    10    10    10    10    10    10
4  58.904300  ...    10    10    10    10    10    10    10    10    10    10

[5 rows x 2091 columns]
```

Visualize Data

In [9]:
```
import pylab, matplotlib
matplotlib.rcParams.update({'font.size': 18})

plt.figure(figsize = (24,8))

for c in labels:
    Xc = X[y == c]
    yc = y[y == c]
    plt.plot(X_names, np.mean(Xc,axis=0).tolist()[0], label= 'Brand' + str(c) +
' (' + str(len(yc)) + ' samples)')
    plt.legend()

plt.title(u'Averaged Spectrums for Each Category\n')
```

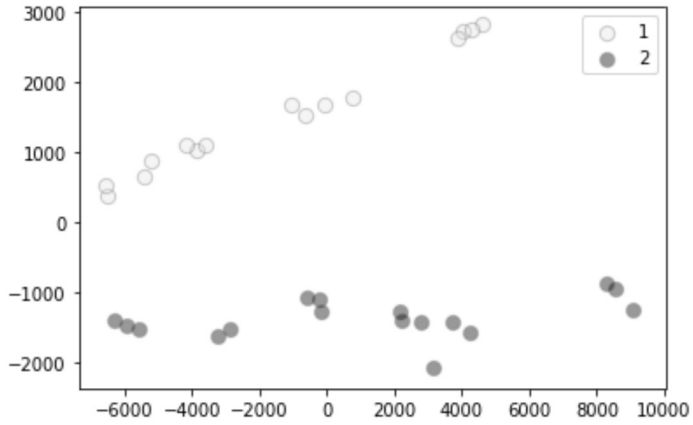
Out[9]:
```
Text(0.5, 1.0, 'Averaged Spectrums for Each Category\n')
```



Plot the first two principal components in the 2D plane

In [11]:
```
# %load plotComponents2D.py
import matplotlib.pyplot as plt
import numpy as np

def plotComponents2D(X, y, labels, use_markers = False, ax=None):

    if X.shape[1] < 2:
        print('ERROR: X MUST HAVE AT LEAST 2 FEATURES/COLUMNS! SKIPPING plotComponents2D().')
        return

        # Gray shades can be given as a string encoding a float in the 0-1 range
    colors = ['0.9', '0.0', '0.5', 'red', 'blue', 'black','orange','green','cyan','purple']
    markers = ['o', 's', '^', 'D', 'H', 'o', 's', '^', 'D', 'H', 'o', 's', '^', 'D', 'H', 'o', 's', '^', 'D', 'H']

    if (ax is None):
        fig, ax = plt.subplots()

    i=0
    for label in labels:
        cluster = X[np.where(y == label)]
        # print(cluster.shape)

        if use_markers:
            ax.scatter([cluster[:,0]], [cluster[:,1]], s=40, marker=markers[i], facecolors='none', edgecolors=colors[i+3],   label=str(label))
        else:
            ax.scatter([cluster[:,0]], [cluster[:,1]], s=70, facecolors=colors[i],  label=str(label), edgecolors = 'gray', alpha = .4) # cmap='tab20'
        i=i+1

    ax.legend()
```

In [12]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2) # keep the first 2 components
pca.fit(X)
X_pca = pca.transform(X)
plotComponents2D(X_pca, y, labels)
```



The two classes have a wide dividing area. After CS, we want to keep this discriminative structure between the two classes.

Generating Sensing Matrix

In [51]:
```
def getSensingMatrix(N, a = 0.2): # N = data set size
    # np.random.seed(10)

    OMEGA = np.zeros((N,N))
    pm = np.random.permutation(np.arange(0, N))
    for i in range(N):
        OMEGA[i, pm[i]] = 1

    PHI = OMEGA[:int(N*a)]

    return PHI, OMEGA
```

In [52]:
```
getSensingMatrix(5, 0.2)
```

Out[52]:
```
(array([[0., 0., 0., 1., 0.]]), array([[0., 0., 0., 1., 0.],
        [1., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1.],
        [0., 0., 1., 0., 0.],
        [0., 1., 0., 0., 0.]]))
```

In [26]:
```
# Another version of sensing /sampling. Without using PHI

def sampler(x, a = 0.2): # N = data set size, a - sampling ratio
    np.random.seed(10)
    N = len(x)
    pm = np.random.permutation(np.arange(1, N))
    r = pm[:int(N*a)]
    xs = x[r]
    return xs
```

In [305]:
```
# the new version of sensing /sampling

def sensing(x, PHI): # x is a list
    assert (x.shape[0] == PHI.shape[1])
    return PHI @ x
```

In [308]:
```
#x = np.random.randn(4)
# x, sampler(x, a = 0.5)

phi, omega = getSensingMatrix(10, 0.2)
x = np.random.randn(10)
x.shape, phi.shape, x, sensing(x, phi)
```

Out[308]:
```
((10,),
 (2, 10),
 array([-1.7639635 ,  0.84521648,  2.03049231, -0.55314797, -0.36917264,
         0.26074389, -1.12290167,  0.55530285, -0.51296706,  2.1098513 ]),
 array([0.84521648, 2.03049231]))
```

Choose Basis Matrix Ψ

Ψ is a `unitary matrix` . i.e. its conjugate transpose is also its inverse, or $\Psi^H = \Psi^{-1}$
For real matrices, unitary is the same as `orthogonal` .

Use 5 candidate basis matrices: IDM (Identity matrix), DCT (discrete cosine transform), DFT (which is widely used in medical imaging), HWT (Hadamard-Walsh transform, which is widely used in signal processing), UDM (uniformly distributed random orthogonal matrix)

IDM doesn't transform into other spaces. It is just for theoretical purpose.

DCT

The function `dctmtx()` generates the n-by-n discrete cosine transform (DCT) matrix.

$$DCT_{M,N} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \sqrt{2}cos(\frac{\pi}{2N}) & \sqrt{2}cos(\frac{3\pi}{2N}) & \cdots & \sqrt{2}cos(\frac{(2N-1)\pi}{2N}) \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{2}cos(\frac{(M-1)\pi}{2N}) & \sqrt{2}cos(\frac{3(M-1)\pi}{2N}) & \cdots & \sqrt{2}cos(\frac{(2N-1)(M-1)\pi}{2N}) \end{bmatrix}$$

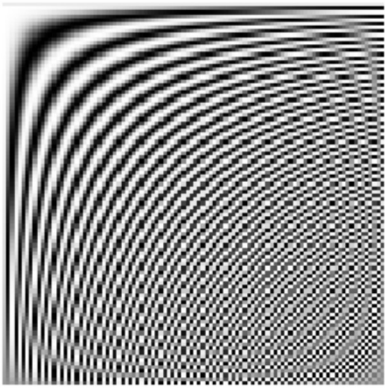
In [6]:
```
import scipy

def dctmtx(m, n):

    mtx = np.zeros((m,n))
    N = n

    mtx[0, :] = 1 * np.sqrt(1/N)
    for i in range(1, m):
        for j in range(n):
            mtx[i, j] = np.cos(np.pi *i * (2*j+1) / (2*N)) * np.sqrt(2/N)

    plt.figure()
    plt.imshow(mtx, interpolation='nearest', cmap=cm.Greys_r)
    plt.axis('off')

    print(np.fabs(scipy.linalg.det(mtx)))

    return mtx
```

Let us generate and visualize a 100x100 DCT matrix.

In [7]:
```
dctmtx(100,100)
```

```
0.9999999999999848
```

Out[7]:
```
array([[ 0.1       ,  0.1       ,  0.1       , ...,  0.1       ,
         0.1       ,  0.1       ],
       [ 0.14140391,  0.14126436,  0.1409854 , ..., -0.1409854 ,
        -0.14126436, -0.14140391],
       [ 0.14135157,  0.14079372,  0.13968022, ...,  0.13968022,
         0.14079372,  0.14135157],
       ...,
       [ 0.00666186, -0.01992644,  0.03301416, ..., -0.03301416,
         0.01992644, -0.00666186],
       [ 0.00444215, -0.01330893,  0.02212317, ...,  0.02212317,
        -0.01330893,  0.00444215],
       [ 0.00222135, -0.00666186,  0.01109579, ..., -0.01109579,
         0.00666186, -0.00222135]])
```



DFT

The function `dftmtx()` generate the n-by-n discrete fourier transform (DFT) matrix

$$DFT_{N,N} = \left(\frac{\omega^{jk}}{\sqrt{N}} \right)_{j,k=0,\ldots,N-1} = \frac{1}{\sqrt{N}} \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \cdots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \cdots & \omega^{N-1} \\ \omega^0 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \omega^0 & \omega^3 & \omega^6 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^0 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

where

$$\omega = e^{-2\pi i/N} = cos(-2\pi/N) + isin(-2\pi/N)$$

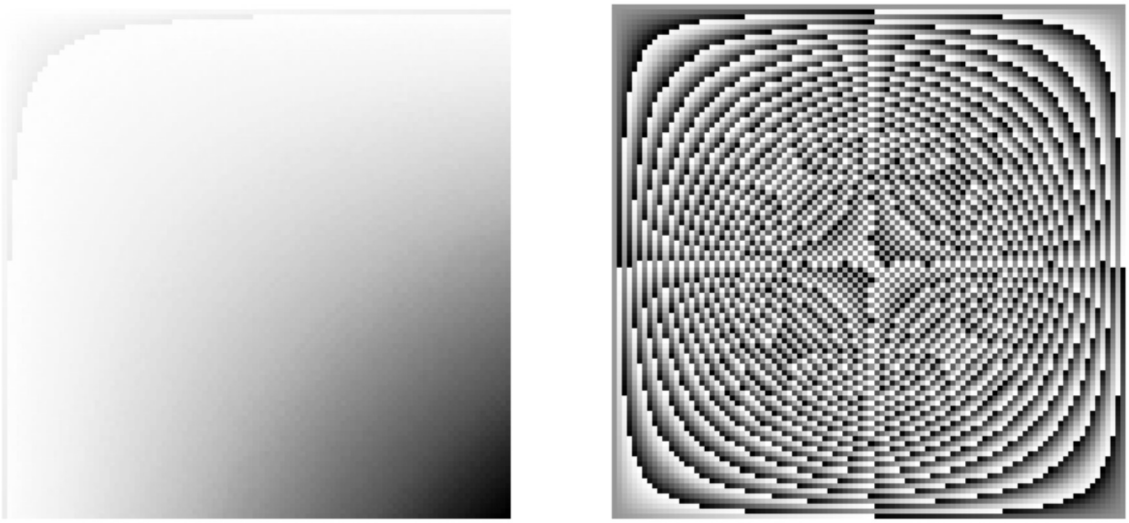
In [8]:
```
import numpy as np
import scipy

def dftmtx(N):
    i, j = np.meshgrid(np.arange(N), np.arange(N))
    w = np.exp( - 2 * np.pi * 1j / N )
    mtx = np.power( w, i * j ) / np.sqrt(N)

    plt.figure(figsize=(12,6))
    # plt.title("abs | phase")
    plt.subplot(1,2,1)
    plt.imshow(abs(mtx), interpolation='nearest', cmap=cm.Greys_r)
    plt.axis('off')
    plt.subplot(1,2,2)
    plt.imshow(np.angle(mtx), interpolation='nearest', cmap=cm.Greys_r)
    plt.axis('off')

    print(scipy.linalg.det(mtx))

    return mtx
```

Let us generate a 100x100 DFT matrix, and visualize the amplitude (abs) and phase (angle) components.

In [9]:
```
dftmtx(100)
```

```
(7.477907182362742e-13+0.9999999999993605j)
```

Out[9]:
```
array([[0.1       +0.j        , 0.1       +0.j        ,
        0.1       +0.j        , ..., 0.1       +0.j        ,
        0.1       +0.j        , 0.1       +0.j        ],
       [0.1       +0.j        , 0.09980267-0.00627905j,
        0.09921147-0.01253332j, ..., 0.09822873+0.01873813j,
        0.09921147+0.01253332j, 0.09980267+0.00627905j],
       [0.1       +0.j        , 0.09921147-0.01253332j,
        0.09685832-0.02486899j, ..., 0.09297765+0.03681246j,
        0.09685832+0.02486899j, 0.09921147+0.01253332j],
       ...,
       [0.1       +0.j        , 0.09822873+0.01873813j,
        0.09297765+0.03681246j, ..., 0.08443279-0.05358268j,
        0.09297765-0.03681246j, 0.09822873-0.01873813j],
       [0.1       +0.j        , 0.09921147+0.01253332j,
        0.09685832+0.02486899j, ..., 0.09297765-0.03681246j,
        0.09685832-0.02486899j, 0.09921147-0.01253332j],
       [0.1       +0.j        , 0.09980267+0.00627905j,
        0.09921147+0.01253332j, ..., 0.09822873-0.01873813j,
        0.09921147-0.01253332j, 0.09980267-0.00627905j]])
```



HWT

The Hadamard-Walsh transform (HWT) matrix requires the dimension to be 2^N.
In function `hwtmtx()` , we will round the input parameter to its next 2^N value.

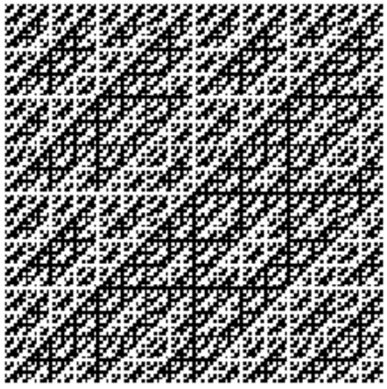
$$HWT_{(2^N)} = \frac{1}{\sqrt{N/2}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & -1 & 1 & -1 & \cdots & 1 & -1 \\ 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & -1 & 1 & -1 & \cdots & 1 & -1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & -1 & 1 & -1 & \cdots & 1 & -1 \end{bmatrix}$$

In [14]:
```
from scipy.linalg import hadamard
import math

def hwtmtx(N):

    NN = 2**(N-1).bit_length() # make sure it is a 2**n value
    print('Expanded to ', NN, ', Divide by', 2**(math.log(NN,2)/2))

    mtx = hadamard(NN) / (2**(math.log(NN,2)/2))

    plt.figure()
    plt.imshow(mtx, interpolation='nearest', cmap=cm.Greys_r)
    plt.axis('off')

    # Use slogdet when mtx is big, e.g. > 2000
    logdet = np.linalg.slogdet(mtx)
    det = logdet[0] * np.exp(logdet[1])
    print('det =', det)

    return mtx, NN
```

In [15]:
```
hwtmtx(100)
```

```
Expanded to  128 , Divide by 11.313708498984761
det = 0.9999999999999938
```

Out[15]:
```
(array([[ 0.08838835,  0.08838835,  0.08838835, ...,  0.08838835,
          0.08838835,  0.08838835],
        [ 0.08838835, -0.08838835,  0.08838835, ..., -0.08838835,
          0.08838835, -0.08838835],
        [ 0.08838835,  0.08838835, -0.08838835, ...,  0.08838835,
         -0.08838835, -0.08838835],
        ...,
        [ 0.08838835, -0.08838835,  0.08838835, ...,  0.08838835,
         -0.08838835,  0.08838835],
        [ 0.08838835,  0.08838835, -0.08838835, ..., -0.08838835,
          0.08838835,  0.08838835],
        [ 0.08838835, -0.08838835, -0.08838835, ...,  0.08838835,
          0.08838835, -0.08838835]]), 128)
```



In the above example, hwtmtx(100) generates a 128x128 matrix. 128 = 2^7

UDM (uniformly distributed random orthogonal matrix)

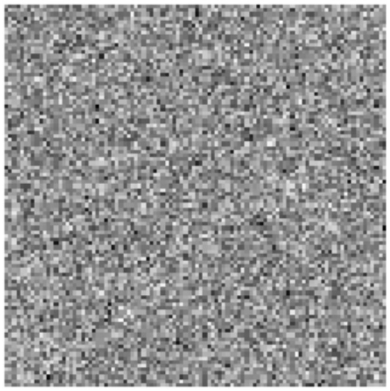
Function `rvsmtx()` generates a uniformly distributed random orthogonal matrix.

In [20]:
```
from scipy.stats import ortho_group

def rvsmtx(N):
    mtx = ortho_group.rvs(N) # uniformly distributed random orthogonal matrix

    plt.figure()
    plt.imshow(mtx, interpolation='nearest', cmap=cm.Greys_r)
    plt.axis('off')

    print(np.fabs(scipy.linalg.det(mtx)))

    return mtx
```

In [21]:
```
rvsmtx(100)
```

```
1.0000000000000044
```

Out[21]:
```
array([[ 0.02662416, -0.14342694, -0.10566114, ..., -0.08841288,
         0.16754891,  0.02157665],
       [-0.17249476,  0.14768207,  0.06745075, ...,  0.11603294,
        -0.00545798,  0.02949385],
       [-0.0241674 , -0.09139529, -0.06255712, ...,  0.05553968,
         0.04324998, -0.00282775],
       ...,
       [-0.00478052,  0.05895622, -0.06395684, ...,  0.27288879,
         0.03907859,  0.02712428],
       [-0.10277921, -0.10599319,  0.0207901 , ...,  0.03194886,
        -0.09970206, -0.00548009],
       [ 0.04876354,  0.06733024, -0.08274201, ...,  0.11713795,
         0.08942622,  0.05651709]])
```



Generate 5 candidate Ψ, and save to PSIS.pkl file.
Next time we can just deserialize the pkl file to load back.

In [ ]:
```
PSIS=[
    np.identity(n), # the identitiy matrix, just for comparison
    dctmtx(n,n),
    dftmtx(n),
    hwtmtx(n)[0],
    rvsmtx(n)
]

import pickle

filehandler = open("PSIS.pkl","wb")
pickle.dump(PSIS,filehandler)
filehandler.close()
```

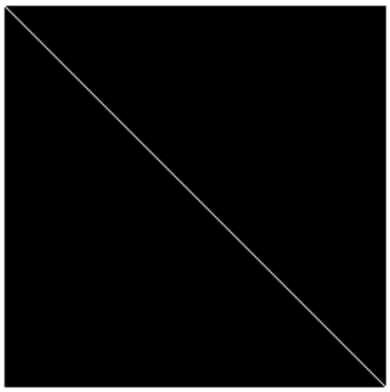
Load PSIs back from pickle

In [22]:
```
# Load PSIs back from pickle

import pickle

file = open("PSIS.pkl",'rb')
PSIS = pickle.load(file)
file.close()
```

Visualize the Ψ candidates

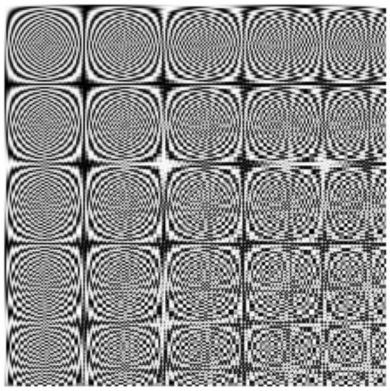
In [25]:
```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline

plt.figure()
plt.imshow(PSIS[0], interpolation='nearest', cmap=cm.Greys_r)
plt.axis('off')
print('IDM')
```

```
IDM
```



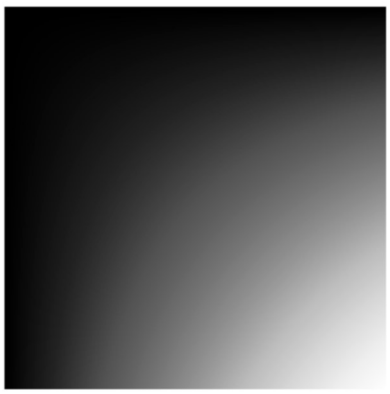
In [27]:
```
plt.figure()
plt.imshow(PSIS[1], interpolation='nearest', cmap=cm.Greys_r)
plt.axis('off')
print('DCT')
```

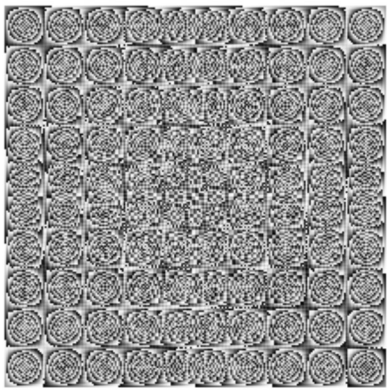
```
DCT
```



In [26]:
```
import numpy as np

plt.figure()
plt.imshow(abs(PSIS[2]), interpolation='nearest', cmap=cm.Greys_r)
plt.axis('off')

plt.figure()
plt.imshow(np.angle(PSIS[2]), interpolation='nearest', cmap=cm.Greys_r)
plt.axis('off')
print('DFT')
```

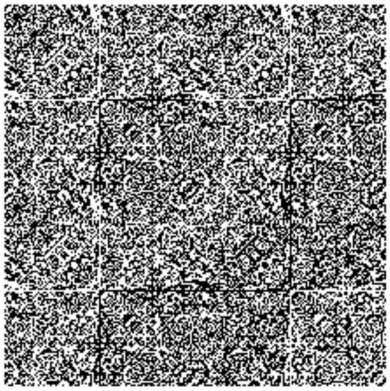
```
DFT
```





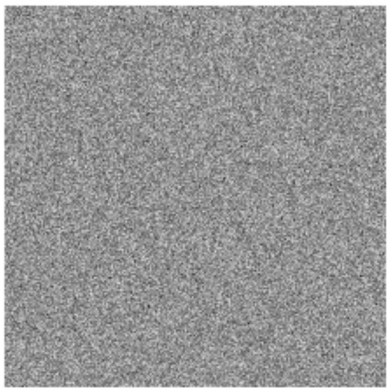
In [32]:
```
plt.figure()
plt.imshow(PSIS[4], interpolation='nearest', cmap=cm.Greys_r)
plt.axis('off')
print('HWT')
```

```
HWT
```



In [33]:
```
plt.figure()
plt.imshow(PSIS[5], interpolation='nearest', cmap=cm.Greys_r)
plt.axis('off')
print('UDM')
```

```
UDM
```



Coherence Between Two Bases

$\mu(A,B) = \sqrt{n} \max_{0 \le i,j < n} (a_i^T b_j)$ a and b are column vectors in A and B

$0 \le \mu(A,B) \le \sqrt{n}$

We are interested in bases pairs with small coherence and in that case we say the two bases are incoherent. In the case of dictionaries for sparse representation, the incoherence is important because a small μ means small correlation between the two bases, thus a dictionary composed by such a pair of bases has a “richer vocabulary” relative to a dictionary with a larger μ .
CS requires a small μ between Ψ and Ω. Ω is a scrambled identity matrix. The sensing matrix Φ is a subset off Ω (take the first few rows).

Theoretical computation results:

> μ(Ω, I) = $\sqrt{n}$
> μ(Ω, C) = $\sqrt{2}$
> μ(Ω, F) = 1
> μ(Ω, H) = 1

$I = IDM, C = DCT, F = DFT, H = HWT$

Function `mutual_coherence()` is defined to calculate μ.
We will check whether the experiment result conforms to the theoretical result.

In [49]:
```
from tqdm import tqdm

def mutual_coherence(A,B):
    assert (A.shape == B.shape)
    assert (A.shape[0] == A.shape[1])
    n = A.shape[0]
    p = 0

    with tqdm(total= n*n) as pbar:
        for i in range(n):
            for j in range(n):
                a = A[:,i].T
                b = B[:,j]
                tmp = a@b
                # print(tmp)
                if (p < tmp):
                    p = tmp
                pbar.update(1)
    return math.sqrt(n)*p
```

In [319]:
```
psi_names = ['Identity Matrix', 'DCT', 'DFT', 'DWT', 'Hadamard-Walsh Matrix', '
Uniformly Distributed Random Orthogonal Matrix']

for idx, PSI in enumerate(PSIS):
    assert (PSI.shape[0] == PSI.shape[1])

    # for some transformations, PSI dimension may differ. e.g. HWT requires 2**
n and DWT requires even number
    n = PSI.shape[0]
    _, OMEGA = getSensingMatrix(n)

    print(psi_names[idx], ": ", mutual_coherence(PSI, OMEGA))
```

```
100%|████████████████████████████████████████████████████████████████| 43
68100/4368100 [03:48<00:00, 19152.96it/s]

Identity Matrix :  45.71651780264984

100%|████████████████████████████████████████████████████████████████| 43
68100/4368100 [03:48<00:00, 19156.82it/s]

DCT :  1.4142135623730951

100%|████████████████████████████████████████████████████████████████| 43
68100/4368100 [03:10<00:00, 22978.39it/s]

DFT :  (1.00000000006146+7.877968100069424e-13j)

100%|████████████████████████████████████████████████████████████████| 1677
7216/16777216 [16:58<00:00, 16469.66it/s]

Hadamard-Walsh Matrix :  1.0

100%|████████████████████████████████████████████████████████████████| 43
68100/4368100 [03:48<00:00, 19138.18it/s]

Uniformly Distributed Random Orthogonal Matrix :  5.293445504102594
```

The result perfectly matches the theoretic calculation.

IDM reaches the upper bound of μ, i.e. $\sqrt{n}$
DCT: $\sqrt{2}$
DFT: 1
Hadamard-Walsh Matrix: 1

Sparsity analysis

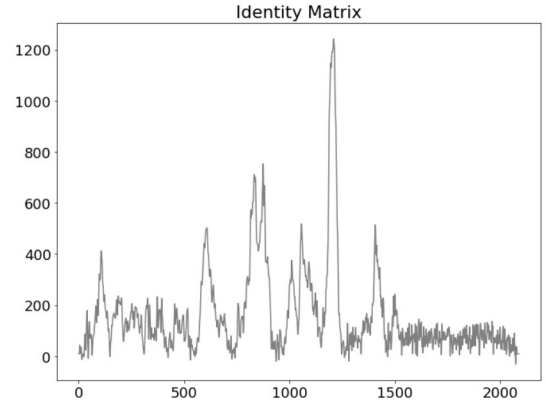
$z = \Psi^H x$

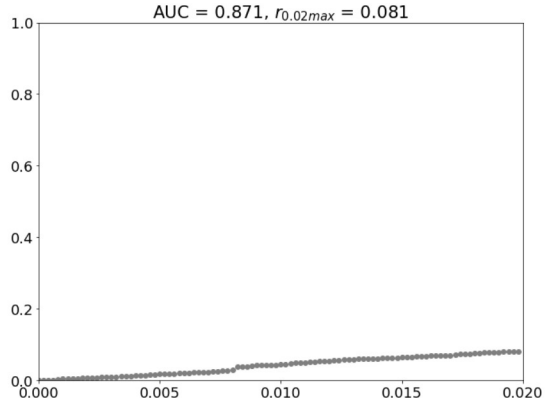
We will transform the original x into the latent space defined by Ψ.

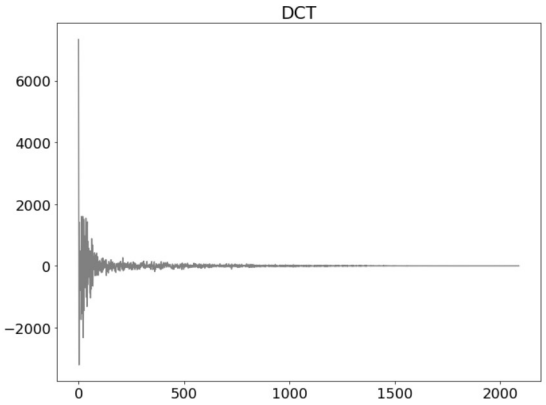
CS requires to choose the Ψ that results in the highest sparsity.

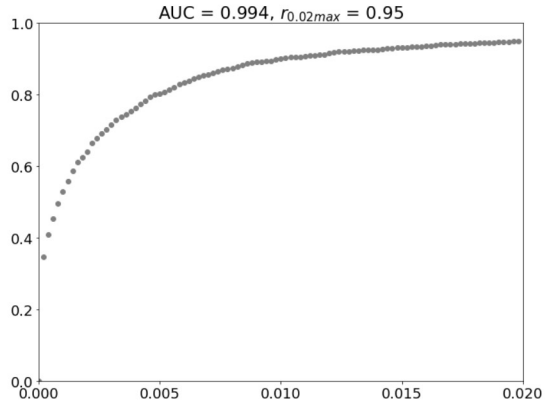
In [601]:
```
x = X[0,:]
x.shape

import pylab, matplotlib

plt.figure(figsize=(24,48))
rows = len(PSIS)
matplotlib.rcParams.update({'font.size': 18})

for idx, PSI in enumerate(PSIS):

    xx = np.copy(x)

    # pad x with zero if necessary
    if (x.shape[1] < PSI.shape[0]):
        xx = np.zeros((x.shape[0], PSI.shape[0]))
        xx[:x.shape[0],:x.shape[1]] = x

    # print(xx.shape, PSI.shape)

    PSI_H = PSI.conj().T
    PSI_INV = np.linalg.pinv(PSI)
    #print(PSI_H)
    #print(PSI_INV)
    # theoretically, PSI_H == PSI_INV
    z = (xx @ PSI_H).ravel().tolist()
    # print(z)
    MAX = abs(max(z, key=abs))
    print(MAX)
    thresholds = np.array(range(100)) / 5000

    rs =[]
    for threshold in thresholds:
        rs.append((np.abs(np.array(z)) <= threshold * MAX).sum() / len(z))

    auc = 0
    for i in range(1000):
        auc += (np.abs(np.array(z)) <= (i+1)/1000 * MAX).sum() / len(z)
    auc = auc/1000

    r = (np.abs(np.array(z)) <= 0.02 * MAX).sum() / len(z)  # use 0.02 MAX ABS
as threshold

    plt.subplot(rows,2,2*idx+1)
    plt.plot(z, color='gray')
    plt.title(psi_names[idx])
    # plt.axis('off')

    plt.subplot(rows,2,2*idx+2)
    plt.scatter(thresholds, rs, color='gray')
    plt.title('AUC = ' + str(round(auc,3)) + ', $r_{0.02max}$ = ' + str(round
(r,3)))
    pylab.xlim([0,0.02])
    pylab.ylim([0,1.0])
    plt.xticks(np.arange(0.0, 0.021, 0.005))
```

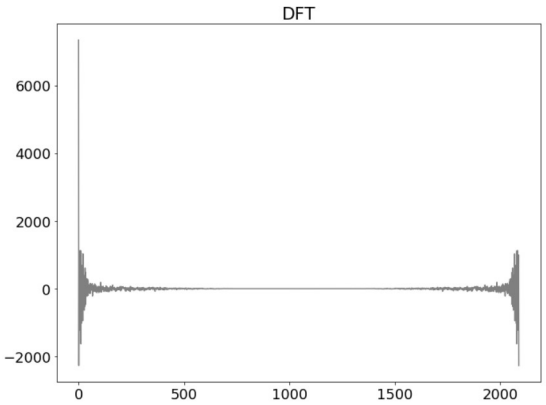
```
1243.019
7338.632439548879
7338.632439548878
5242.136258902184
845.6028616661284
```

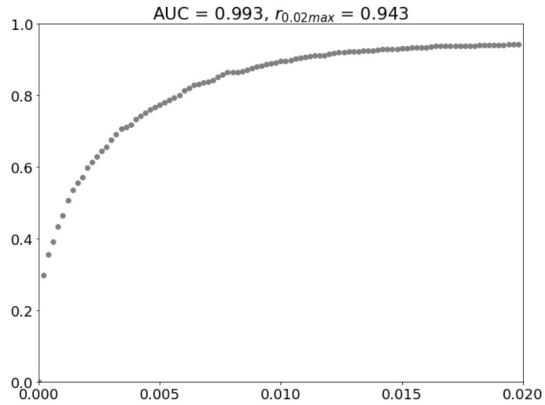

Identity Matrix
1200
1000
800
600
400
200
0
0
500
1000
1500
2000

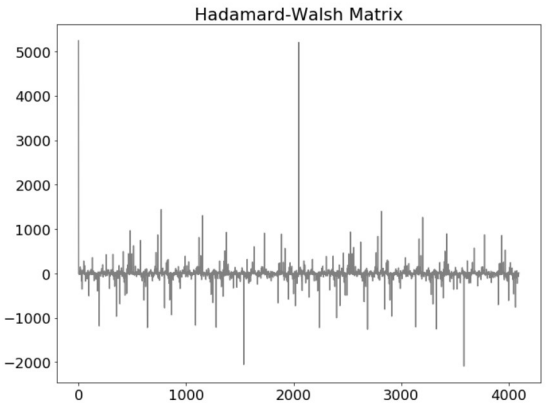

AUC = 0.871, $r_{0.02max}$ = 0.081
1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.005
0.010
0.015
0.020

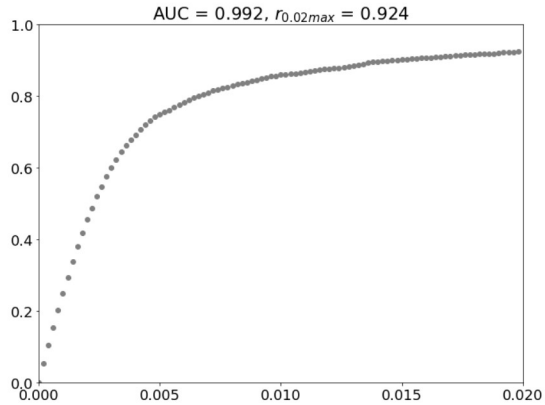

DCT
6000
4000
2000
0
−2000
0
500
1000
1500
2000

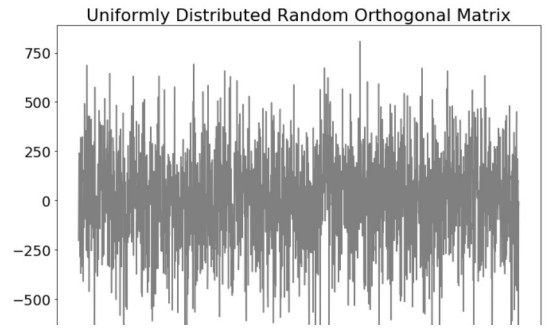

AUC = 0.994, $r_{0.02max}$ = 0.95
1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.005
0.010
0.015
0.020

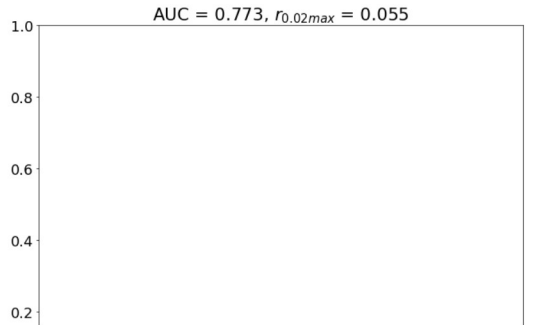

DFT
6000
4000
2000
0
−2000
0
500
1000
1500
2000


AUC = 0.993, $r_{0.02max}$ = 0.943
1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.005
0.010
0.015
0.020


Hadamard-Walsh Matrix
5000
4000
3000
2000
1000
0
−1000
−2000
0
1000
2000
3000
4000


AUC = 0.992, $r_{0.02max}$ = 0.924
1.0
0.8
0.6
0.4
0.2
0.0
0.000
0.005
0.010
0.015
0.020


Uniformly Distributed Random Orthogonal Matrix
750
500
250
0
−250
−500


AUC = 0.773, $r_{0.02max}$ = 0.055
1.0
0.8
0.6
0.4
0.2

Best AUC: DFT DCT
Best μ: DFT HWT

Finally, Choose DFT

Under DFT, at the 0.02 max abs threshold, non-sparsity degree d ≈ 1 - 0.95 = 0.05. According to the "four-to-one" practical / empirical rule: for exact reconstruction, one needs about `four incoherent measurements per unknown nonzero term`

k >= 4 d (d is percentage of non-zeros in the signal, 1 - degree of sparsity)
k >= 0.2, i.e. 0.2 is the lower bound for effective CS.

Sensing Matrix for sub-Nyquist sampling

batch process by vectorization

In [348]:
```
k = 0.2
PHI, OMEGA = getSensingMatrix(X.shape[1], k)

PHI.shape, OMEGA.shape
```

Out[348]:
```
((418, 2090), (2090, 2090))
```

In [349]:
```
Xs = PHI @ X.T
Xs.shape
```

Out[349]:
```
(418, 46)
```

single-sample processing

In [55]:
```
import numpy as np

def sampler(x, k = 0.2): # N = data set size, a - sampling ratio

    N = len(x)

    # when k > 1, return x directly
    if (k > 1):
        return x, list(range(N))

    pm = np.random.permutation(np.arange(1, N))
    idx = pm[:int(N*k)]
    xs = np.array(x)[idx]

    return xs, idx
```

In [387]:
```
x = X[0,:].ravel().tolist()[0] # get the first sample
xs, idx = sampler(x, k = 0.2)
xs.shape, idx
```

Out[387]:
```
((418,),
 array([1105, 1875,  539,  141, 1973, 1071, 1048,  577,  783,    5, 1753,
        1687, 1133, 1950, 1443,  808, 1634, 2085,  928, 1879, 1683, 1736,
        1448, 1888, 2038, 1178, 1053,  460, 2042, 1150,  992,  462,  919,
         405, 1198, 1589,  862, 1721,  768,  731, 1580, 1974,  107,  644,
        1077,  274,   28, 1792, 1242,  263, 1026, 1598, 1394,  293, 1819,
        2004, 1136,  259,  642, 1060,  629,  612, 1384, 2039,    9,  192,
        1309, 1926,  203, 1931, 1465, 1109, 1392, 1829,  334, 1254, 1887,
        1399,  907, 1708,  195, 1203, 1821,  483, 1230,  247,  340,  690,
         181, 1986,  261,   27,  666,  354, 1115, 1741,  682,  601,  296,
        1776,   39, 1604, 1345, 1939,   89, 1415, 1471,  110, 1353, 1566,
         498,  574, 1765,   98,  760, 1251, 1014,  212, 1482, 1218, 1913,
        1354, 2028,  670,  278,   14, 1565,  356, 1696,    8, 1045, 1749,
         774,  267, 1331, 1219,  943,  708,  871,  991, 1395,  667, 1393,
         464,   38, 1861, 1369, 1293,  843, 1490,  398,  132, 1632,  924,
         297,  438,   57,   16,  540, 1593, 1344,  338, 1912,   73, 1599,
         108,  791,  439,  359, 1483,  990, 1356, 1576,  578,  741, 1185,
         517, 1426,  139, 1489, 1383, 1248,  295, 1911,  446, 1874,  163,
         993, 1363,  222, 1271,  428, 2009,   81,  710, 1481,  766, 2082,
         199, 1241,  288, 1863,  727,  887,  200,   53, 1575,  117,  223,
        1153, 1275, 2019,  492, 1664,  789, 1731,  207,  273,  479, 1716,
        1257,  635, 1584, 1449, 1639,  482, 1247, 1166, 1480, 1617, 1520,
         672,  976, 1928,  905, 1364, 1311,  893,  812, 1917, 1260,   55,
         883, 1336, 1312, 1085,  179, 1339, 1704,  617, 1027, 1952,  845,
         586,  360,  282, 1327,  941, 1552,   37,  499, 1050, 2065, 1463,
         129, 1979, 2035,  140,  759,  451,  720, 1243,  643,  886, 1496,
          35,  916,  949, 1387,  403,  374, 1236, 1341, 1146, 1597,  765,
        1838, 1557, 1603, 1088, 1832, 1036, 1194, 1944,  501, 1120,  364,
        1648, 1854,  610,  685,  175,  563, 1382,  535, 1497,  614,  312,
        1375,  860,   36,  399,  333, 1209,  101,   51, 1815,  281,  728,
         477,  246,   82, 1823,  951,  372, 1074, 1684, 1529, 1372, 1608,
         528,  299, 1102, 1359,  944, 1546, 2023,   30, 1763, 1280, 1714,
          64, 1942,  136, 1017, 1197, 1056, 1335,  835,  343, 1184, 1833,
        1154, 2048,  434, 1717, 1798,  233,  300, 1170, 1880,  622, 1945,
        1622, 1836,  604, 1706,  549,  541, 1447, 1641,  668, 1761, 1637,
         693, 1222, 1454, 1978, 1021,  347,  230,  750, 1234, 1029, 1971,
        1065,   61,  821,  600,  489, 1627,  816, 1545,  189, 1245, 2079,
         313,  840, 1672,  465, 1610,  769, 1317,  275,  873, 2016, 1667,
        1445, 1286,  271, 1516,  879, 1840, 1884, 1573, 1340,  908, 1411]))
```

From the original 2090-point signal, only 418 points are sampled.

Measuring Matrix $A = \Phi\Psi$

In [361]:
```
# Get the PSI for DFT
PSI = PSIS[1]
```

In [366]:
```
A = PHI @ PSI
A.shape, PHI.shape, PSI.shape, Xs.shape
```

Out[366]:
```
((418, 2090), (418, 2090), (2090, 2090), (418, 46))
```

can also use the following to get the measuring matrix A

In [3]:
```
import cv2

def csmtx(N, idx, t = 'dct'):

    K = len(idx)
    a = np.zeros((1, N))

    # Suppose the first number in the permutation is 42. Then choose the 42th
    # elements in a matrix full of zeros and change it to 1. Apply a discrete
    # cosine transformation to this matrix, and add the result to other matrix
    # Adelta as a new row.
    A = np.zeros((K, N))
    for i, j in enumerate(idx):
        a[0,j] = 1
        if t == 'dft':
            A[i, :] = cv2.dft(a)
        else:
            A[i, :] = cv2.dct(a)
        a[0,j] = 0

    ###### ALTERNATIVE IMPLEMENTATION ######
    # phi = np.zeros((K, N)) # Sampling matrix, kxN
    # for i, j in enumerate(r):
    #    phi[i,j] = 1
    # psi = dctmtx(N,N) # memory costly
    # A = phi*psi

    return A
```

In [391]:
```
A = csmtx(len(x), idx, 'dft')
A.shape
```

Out[391]: (418, 2090)

L1 Minimization & Reconstruction

Let $A = \Phi\Psi$,

$x_s = Az$

$A = [a_1, a_2, \ldots, a_n]$

Each a is column vector / atom / dictionary entry,

Because A has more columns than rows, A is `overcomplete` . That means A has excessive dictionary entries (non-repetive), and can express data with high sparsity.

A is also `full row rank` , i.e. rank(A) = r (row number)

The equations specified by $x_s = Az$ has many (not unique) solutions. This is an `underdetermined linear system` that has more unknowns than equations.
The solutions can be represented as $z = A^T(AA^T)^{-1}x_s + V[r:,:]v$, $A = USV^T$, v is a freely chosen (n-r) × 1 vector .

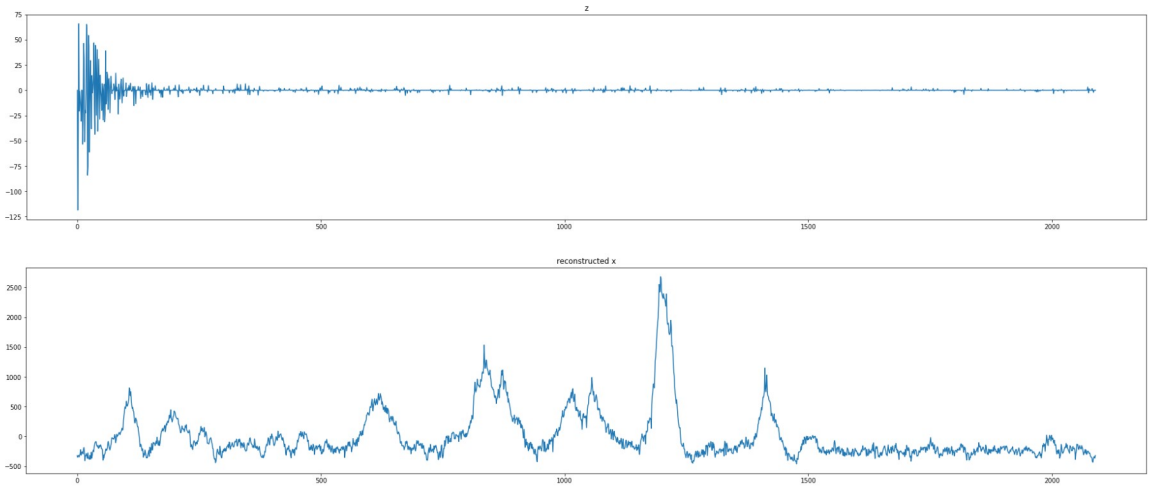
We can find the sparsist solution by minimize L1-norm or $||z||_1$. We don't use L0-norm because minimizing it is `NP hard` .

> objective: minimize ||z||
> subject to: $x_s = Az$

In [49]:
```
from sklearn.linear_model import Lasso
import cv2
import matplotlib

def lasso_cs(A, xs, L1, t = 'dct', silent = 'True'):
    lasso = Lasso(alpha=L1)
    lasso.fit(A,xs)
    # print ('non-zero coefs: ', np.count_nonzero(lasso.coef_))

    if t=='dft':
        x_r = cv2.idft(lasso.coef_)
    else:
        x_r = cv2.idct(lasso.coef_)

    if (silent == False):
        matplotlib.rcParams.update({'font.size': 20})
        fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(32,6))
        ax[0].plot(lasso.coef_, color = 'gray')
        #ax[0].title.set_text('z')

        ax[1].plot(x_r, color = 'gray')
        #ax[1].title.set_text('reconstructed x')

        fig.tight_layout()
        plt.show()

    return x_r, lasso.coef_
```

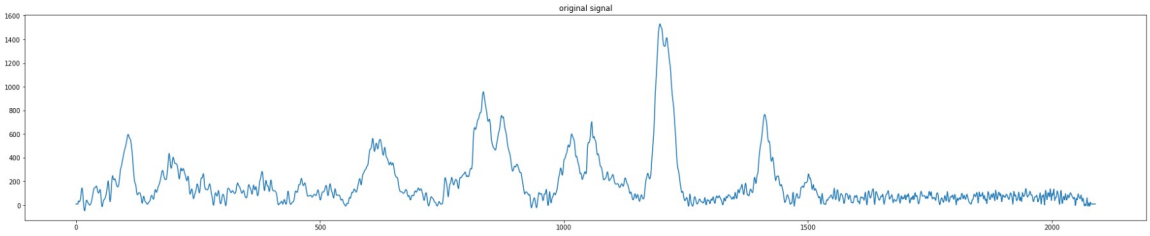
In [408]:
```
xr, z = lasso_cs(A, xs, 0.005, 'dft', silent = False)
```

```
non-zero coefs:  621
```



In [410]:
```
plt.figure(figsize=(32,6))
plt.plot(x)
plt.title('original signal')
```

Out[410]:
```
Text(0.5, 1.0, 'original signal')
```



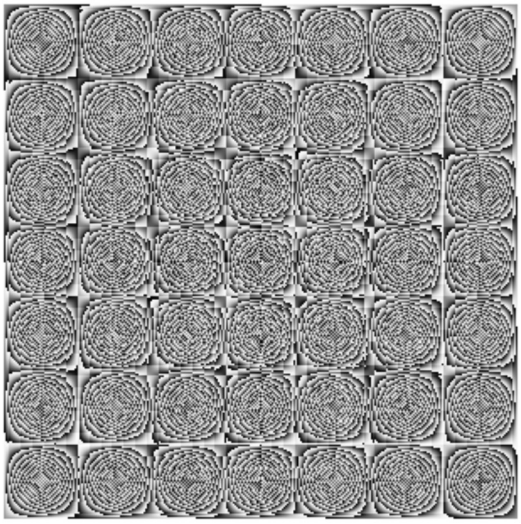
Let us put all the above functions together

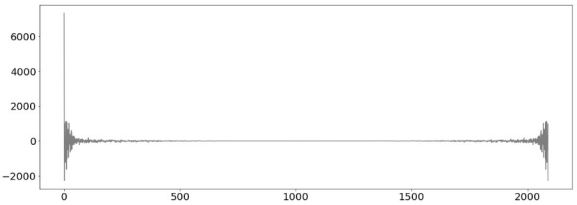
In [56]:
```
def one_sample_CS(x, k = 0.2, L1 = 0.002, t='dft', silent = True):

    xs, idx = sampler(x, k)
    A = csmtx(len(x), idx, t)

    xr, z = lasso_cs(A, xs, L1 = L1, t = t, silent = silent)

    return xr, z
```

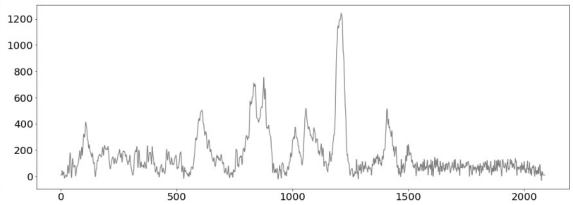
In [61]:
```
x = X[0,:].ravel().tolist()[0]

PSI_H = dftmtx(len(x)).conj().T
z = (x @ PSI_H).ravel().tolist()

### Draw the original x ###
matplotlib.rcParams.update({'font.size': 20})
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(32,6))
ax[0].plot(z, color = 'gray')
#ax[0].title.set_text('z')

ax[1].plot(x, color = 'gray')
#ax[1].title.set_text('reconstructed x')

fig.tight_layout()
plt.show()


for k in [0.1, 0.2, 0.3, 0.4, 1.1]:
    for L1 in [0, 0.001, 0.01, 0.1, 1, 10]:
        print('===== k =', k, ', L1 =', L1,' ======')
        one_sample_CS(x, k = k, L1 = L1, t='dft', silent = False)
```

```
(-1.0000000335595836-1.247326686382166 4e-10j)
```

```
C:\Users\eleve\Anaconda3\lib\site-packages\numpy\core\numeric.py:538: ComplexW
arning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

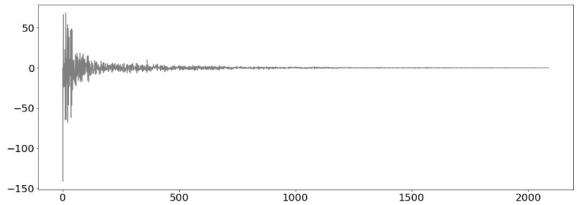


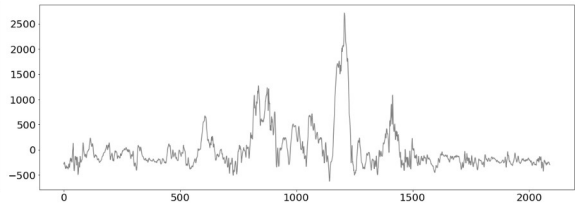




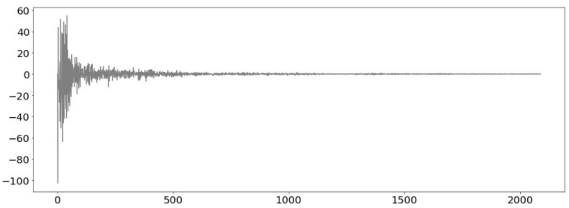

```
===== k = 0.1 , L1 = 0  ======
```

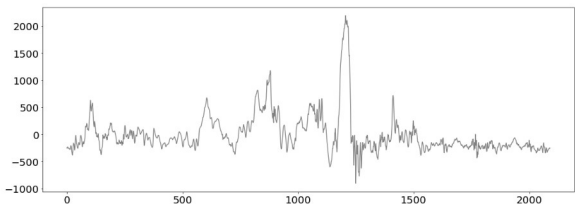
```
C:\Users\eleve\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: UserWarnin
g: With alpha=0, this algorithm does not converge well. You are advised to use
the LinearRegression estimator
  import sys
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: UserWarning: Coordinate descent with no regularization may lead t
o unexpected results and is discouraged.
  positive)
```





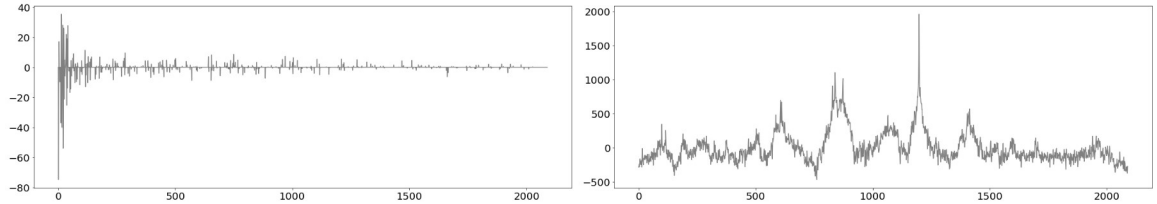

```
===== k = 0.1 , L1 = 0.001  ======
```





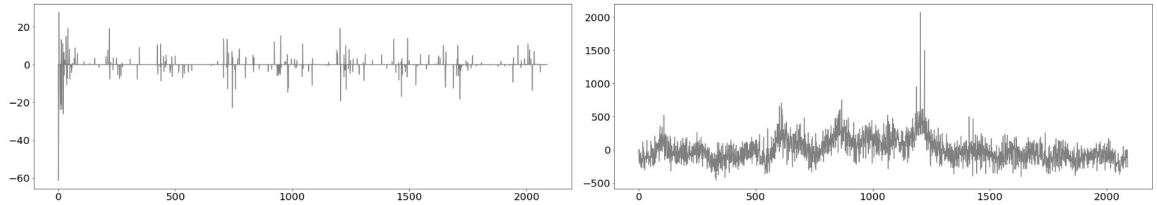

```
===== k = 0.1 , L1 = 0.01  ======
```

```
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 948.060011874864, tolerance: 4
41.08025135244895
  positive)
```

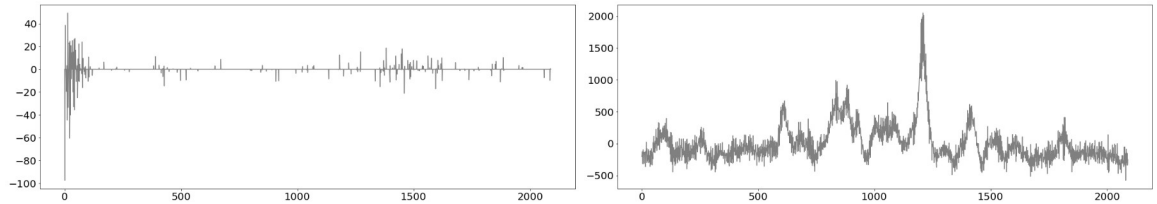


```
===== k = 0.1 , L1 = 0.1  ======
```

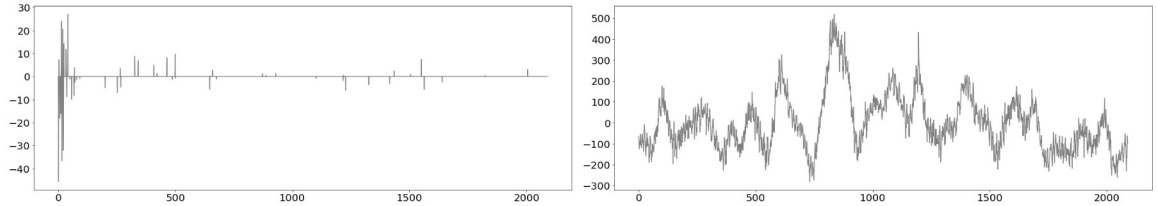
```
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 905.6630159714474, tolerance:
423.1507963585478
  positive)
```



```
===== k = 0.1 , L1 = 1  ======
```

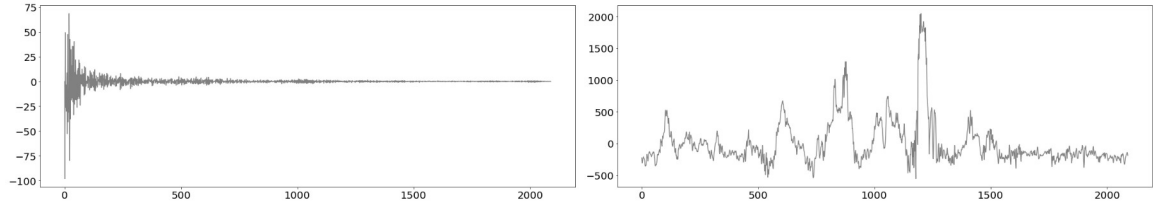


```
===== k = 0.1 , L1 = 10  ======
```

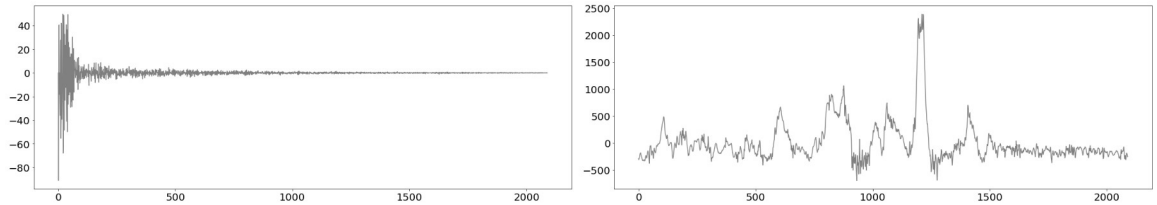


```
===== k = 0.2 , L1 = 0  ======
```

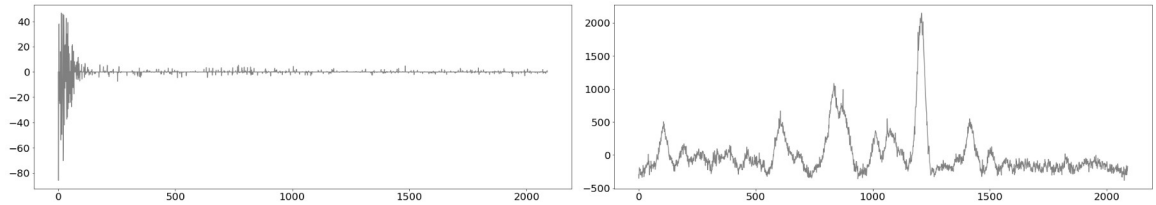
```
C:\Users\eleve\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: UserWarnin
g: With alpha=0, this algorithm does not converge well. You are advised to use
the LinearRegression estimator
  import sys
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: UserWarning: Coordinate descent with no regularization may lead t
o unexpected results and is discouraged.
  positive)
```



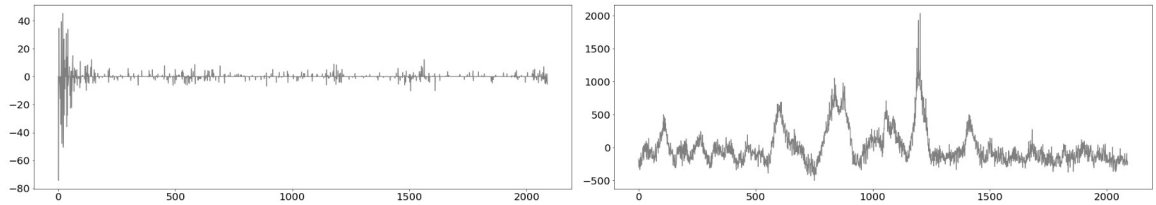
```
===== k = 0.2 , L1 = 0.001  ======
```



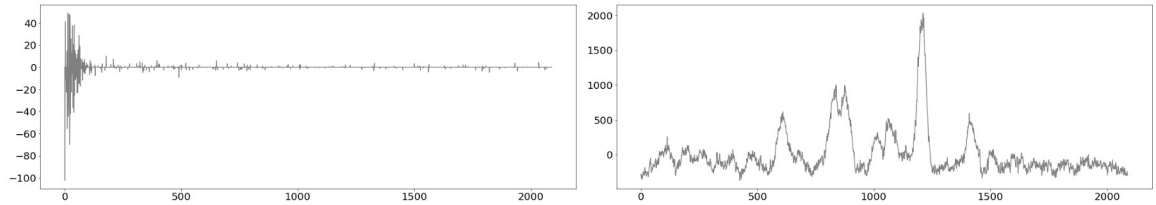
```
===== k = 0.2 , L1 = 0.01  ======
```



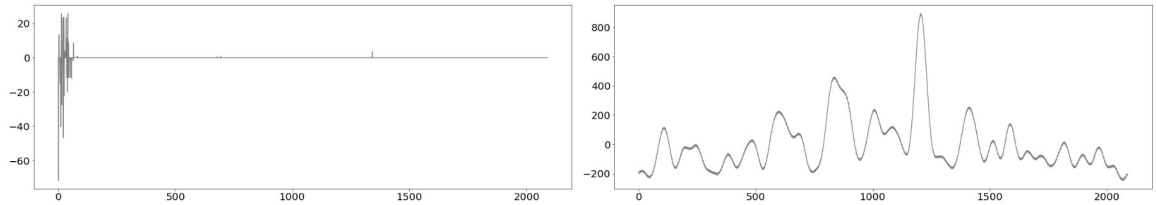
```
===== k = 0.2 , L1 = 0.1  ======
```



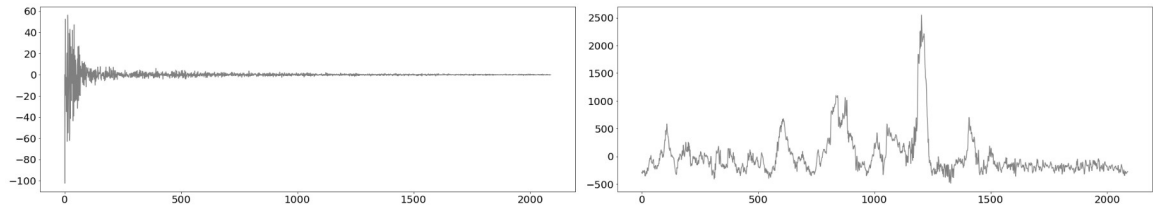
```
===== k = 0.2 , L1 = 1  ======
```



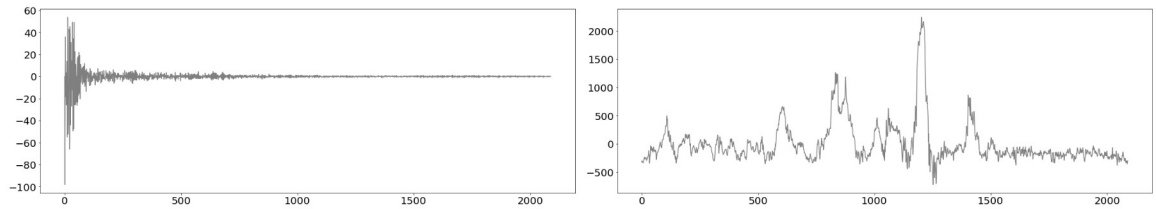
```
===== k = 0.2 , L1 = 10  ======
```



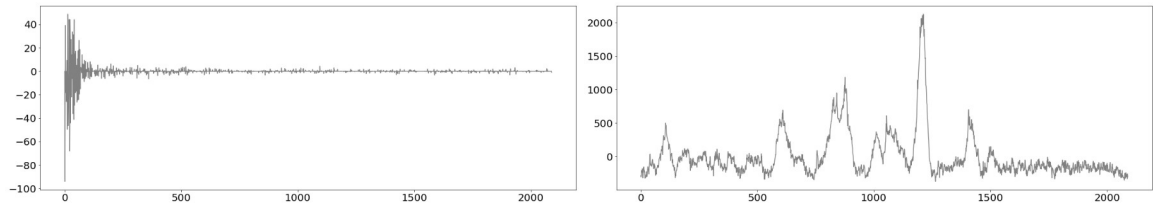
```
===== k = 0.3 , L1 = 0  ======

C:\Users\eleve\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: UserWarnin
g: With alpha=0, this algorithm does not converge well. You are advised to use
the LinearRegression estimator
  import sys
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: UserWarning: Coordinate descent with no regularization may lead t
o unexpected results and is discouraged.
  positive)
```



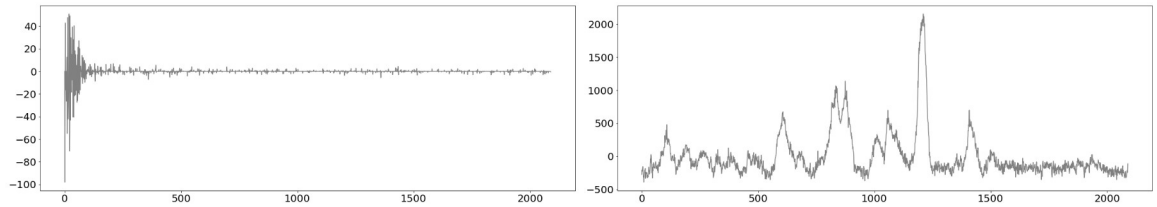
```
===== k = 0.3 , L1 = 0.001  ======
```



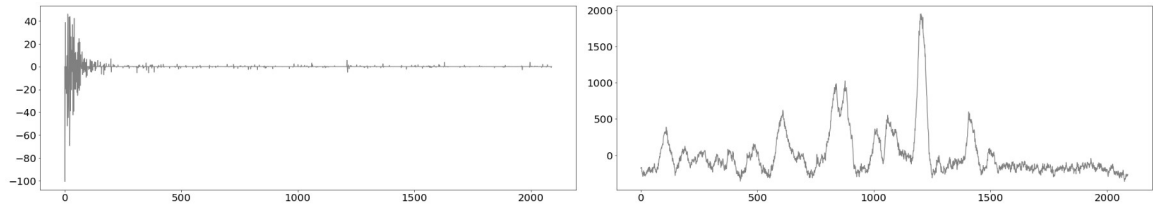
```
===== k = 0.3 , L1 = 0.01  ======
```



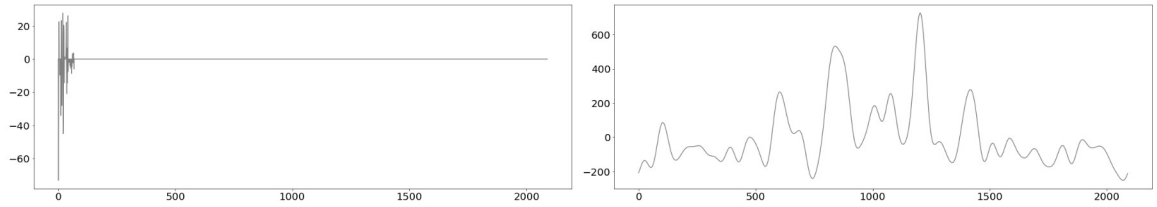
```
===== k = 0.3 , L1 = 0.1  ======
```



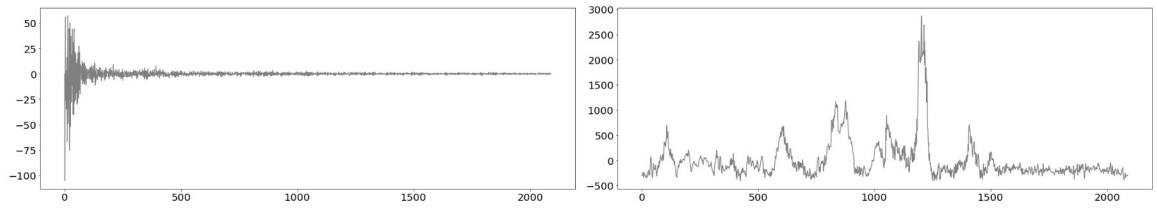
```
===== k = 0.3 , L1 = 1  ======
```



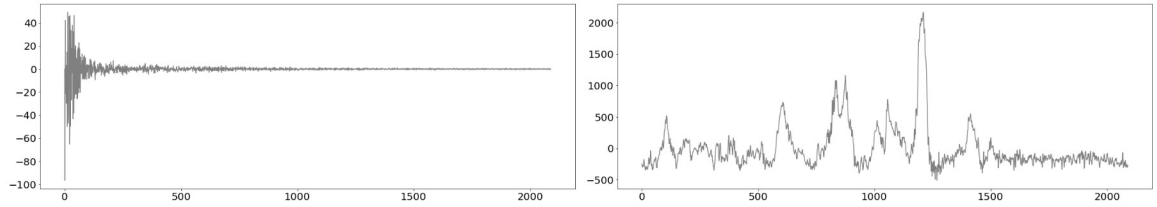
```
===== k = 0.3 , L1 = 10  ======
```



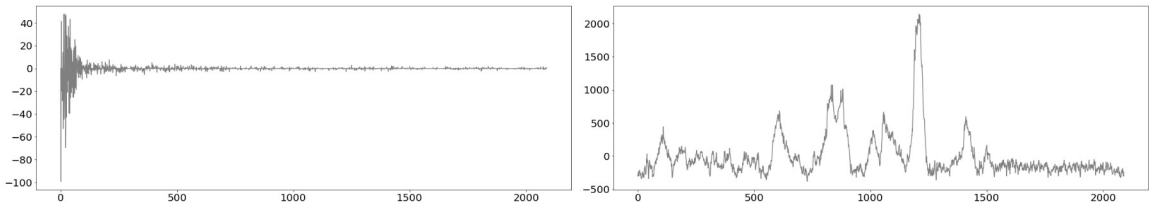
```
===== k = 0.4 , L1 = 0  ======

C:\Users\eleve\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: UserWarnin
g: With alpha=0, this algorithm does not converge well. You are advised to use
the LinearRegression estimator
  import sys
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: UserWarning: Coordinate descent with no regularization may lead t
o unexpected results and is discouraged.
  positive)
```



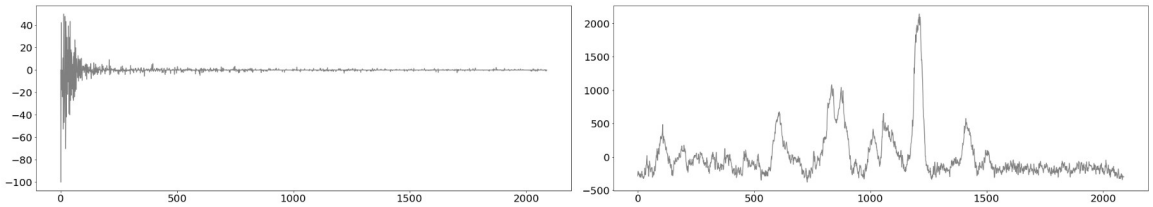
```
===== k = 0.4 , L1 = 0.001  ======
```



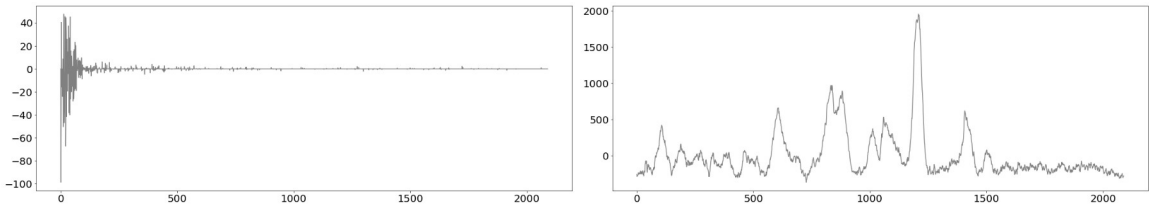
```
===== k = 0.4 , L1 = 0.01  ======
```



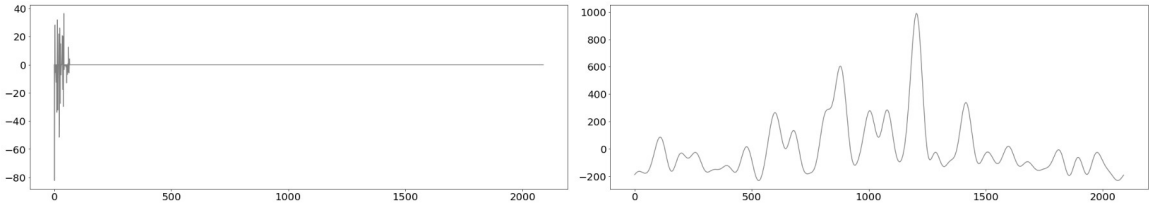
```
===== k = 0.4 , L1 = 0.1  ======
```



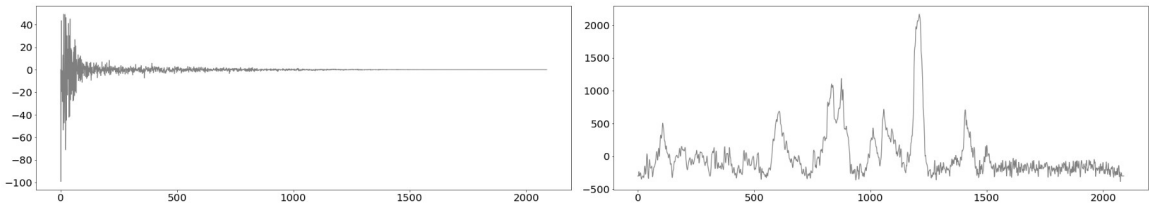
```
===== k = 0.4 , L1 = 1  ======
```



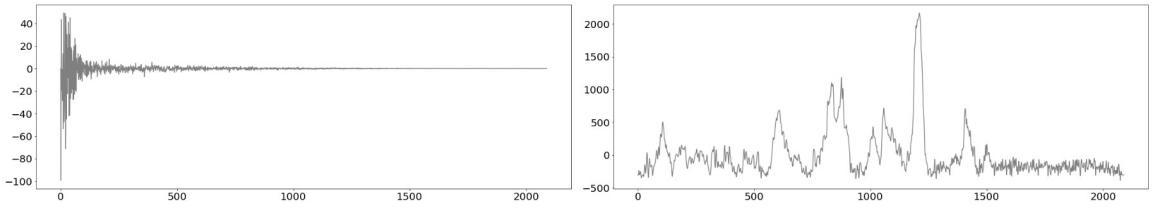
```
===== k = 0.4 , L1 = 10  ======
```



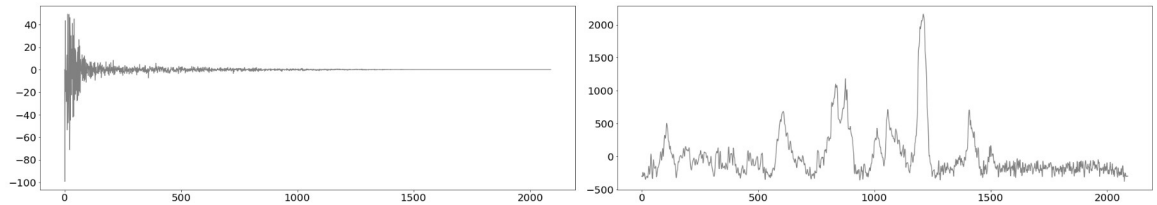
```
===== k = 1.1 , L1 = 0  ======

C:\Users\eleve\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: UserWarnin
g: With alpha=0, this algorithm does not converge well. You are advised to use
the LinearRegression estimator
  import sys
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: UserWarning: Coordinate descent with no regularization may lead t
o unexpected results and is discouraged.
  positive)
```



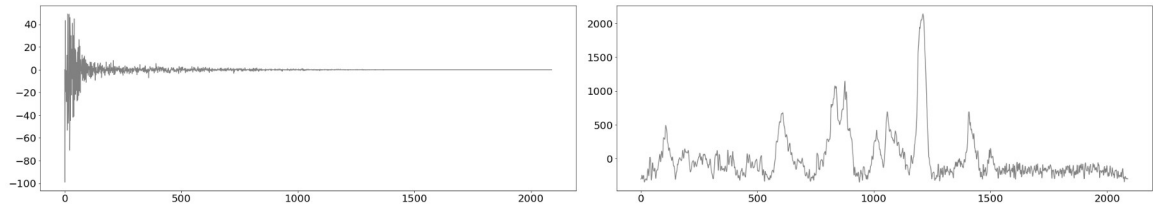
```
===== k = 1.1 , L1 = 0.001  ======
```



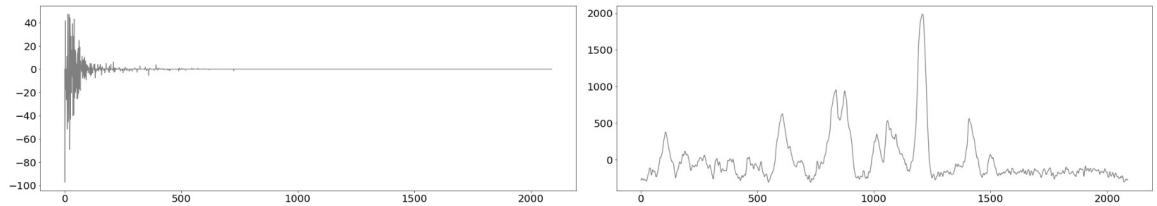
```
===== k = 1.1 , L1 = 0.01  ======
```



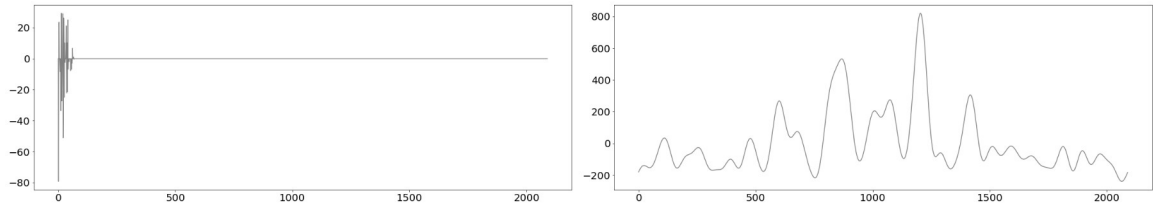
```
====== k = 1.1 , L1 = 0.1  ======
```



```
====== k = 1.1 , L1 = 1  ======
```



```
====== k = 1.1 , L1 = 10  ======
```



In [29]:
```
import scipy

def all_samples_CS (X, y, k, L1, t='dft'):

    print('===== k =', k, ', L1 =', L1,' ======')

    Z = np.zeros(X.shape)
    Xr = np.zeros(X.shape)

    if (k > 1.0): # when k > 1.0, return original signal directly
        Xr = X
        Z = X
    else:
        for i in range(X.shape[0]):
            x = X[i,:].ravel().tolist()[0] # get the first sample
            xr, z = one_sample_CS(x, k, L1, t)
            Z[i,:] = z
            Xr[i,:]= xr[:,0]


    fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20,4))

    from sklearn.decomposition import PCA

    pca = PCA(n_components=2) # keep the first 2 components
    Z_pca = pca.fit_transform(Z)
    # plotComponents2D(Z_pca, y, labels, use_markers = False, ax=ax[0])
    # ax[0].title.set_text('2D-Visualization')

    pca = PCA(n_components=2) # keep the first 2 components
    Xr_pca = pca.fit_transform(Xr)
    plotComponents2D(Xr_pca, y, labels, use_markers = False, ax=ax[0])
    ax[0].title.set_text('2D Visualization')

    Xc1s = []
    Xc2s = []
    title = ''
    for c in set(y):
        # print(Xr.shape, Z.shape, Xr_pca.shape, Z_pca.shape, y.shape)
        Xc = Xr_pca[y == c]
        yc = y[y == c]
        Xc1s.append(list(np.asarray(Xc[:,0]).reshape(1,-1)[0])) # First PC of Cl
ass c
        Xc2s.append(list(np.asarray(Xc[:,1]).reshape(1,-1)[0])) # Second PC of C
lass c

    #### ANOVA ####

    ax[1].title.set_text('PC1')
    ax[1].boxplot(Xc1s, notch=False) # plot 1st PC of all classes
    f,p= scipy.stats.f_oneway(Xc1s[0], Xc1s[1]) # equal to ttest_ind() in case o
f 2 groups
    print("f={},p={}".format(f,p))
    # dict_anova_p1[idx] = p

    ax[2].title.set_text('PC2')
    ax[2].boxplot(Xc2s, notch=False)
    f,p= scipy.stats.f_oneway(Xc2s[0], Xc2s[1])
    print("f={},p={}".format(f,p))

    #### MANOVA ####

    import statsmodels
    from statsmodels.base.model import Model
    from statsmodels.multivariate.manova import MANOVA

    df = pd.DataFrame({'PC1':Xr_pca[:,0],'PC2':Xr_pca[:,1],'y':y})
```

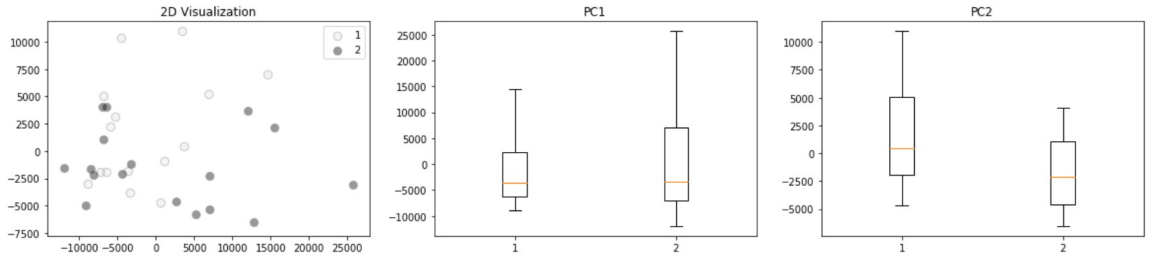
NOTE:

`fig, axs = plt.subplots(3)` returns a 1D array of subplots.

`fig, axs = plt.subplots(3, 2)` returns a 2D array of subplots.

In [35]:
```
result = {}

for k in [0.1, 0.2, 0.3, 0.4]:
    for L1 in [0.001, 0.01, 0.1, 1, 10]:
        acc, manova = all_samples_CS (X, y, k=k, L1=L1)
        result['k = ' + str(k) + 'L1 = ' + str(L1)] = (acc, manova)
```

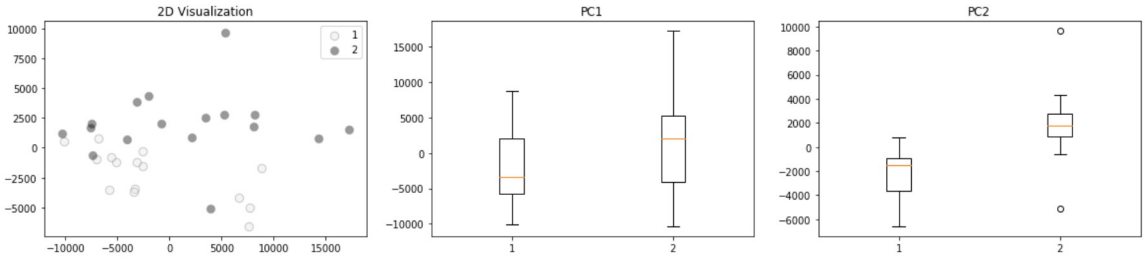
```
===== k = 0.1 , L1 = 0.001  ======
f=0.7493537517132374,p=0.3935511106932027
f=4.77386574367189,p=0.036840699804596234
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
============================================================

------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.8516 2.0000 29.0000  2.5275 0.0973
         Pillai's trace 0.1484 2.0000 29.0000  2.5275 0.0973
 Hotelling-Lawley trace 0.1743 2.0000 29.0000  2.5275 0.0973
    Roy's greatest root 0.1743 2.0000 29.0000  2.5275 0.0973
------------------------------------------------------------

------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.8383 2.0000 29.0000  2.7959 0.0776
         Pillai's trace 0.1617 2.0000 29.0000  2.7959 0.0776
 Hotelling-Lawley trace 0.1928 2.0000 29.0000  2.7959 0.0776
    Roy's greatest root 0.1928 2.0000 29.0000  2.7959 0.0776
============================================================

SVC ACC:  0.7827380952380952
```




```
===== k = 0.1 , L1 = 0.01  ======
```

```
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 980.9499725346309, tolerance:
782.986769950874
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 995.5335269016138, tolerance:
512.2348822399686
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1543.764280724276, tolerance:
1250.5539716828796
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1671.1089656092902, tolerance:
1041.9075147361805
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1192.4934233118297, tolerance:
892.2243608059954
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1347.4624049441086, tolerance:
845.7480385807359
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1172.0226256172343, tolerance:
813.8664765105298
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2188.622850741333, tolerance:
1209.561852475047
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1739.7490038627252, tolerance:
1047.1934162250868
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 962.6899210992066, tolerance:
569.8125551071149
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1394.6425179301575, tolerance:
792.0628916975884
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1804.9471506161372, tolerance:
1075.5320194624458
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1998.4865673491176, tolerance:
1595.4903001871337
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2632.7692895190144, tolerance:
2083.5362398239154
```

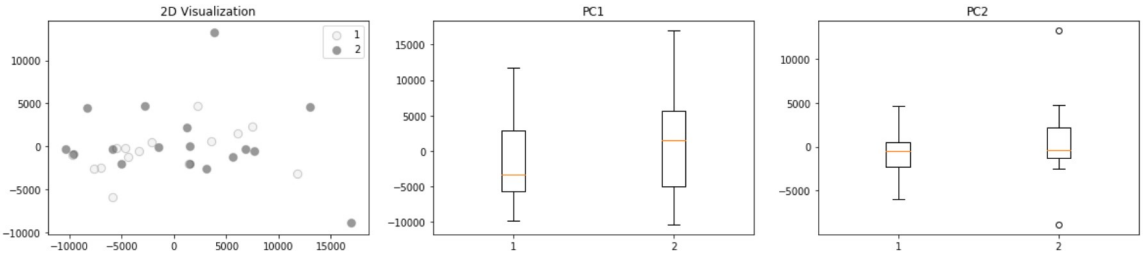
```
f=1.5432129009089282,p=0.22376380234100307
f=20.856841878700727,p=7.890254781126888e-05
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
============================================================

------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.5659 2.0000 29.0000 11.1226 0.0003
         Pillai's trace 0.4341 2.0000 29.0000 11.1226 0.0003
 Hotelling-Lawley trace 0.7671 2.0000 29.0000 11.1226 0.0003
    Roy's greatest root 0.7671 2.0000 29.0000 11.1226 0.0003
------------------------------------------------------------

------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.5410 2.0000 29.0000 12.3038 0.0001
         Pillai's trace 0.4590 2.0000 29.0000 12.3038 0.0001
 Hotelling-Lawley trace 0.8485 2.0000 29.0000 12.3038 0.0001
    Roy's greatest root 0.8485 2.0000 29.0000 12.3038 0.0001
============================================================

SVC ACC:  0.8809523809523809
```



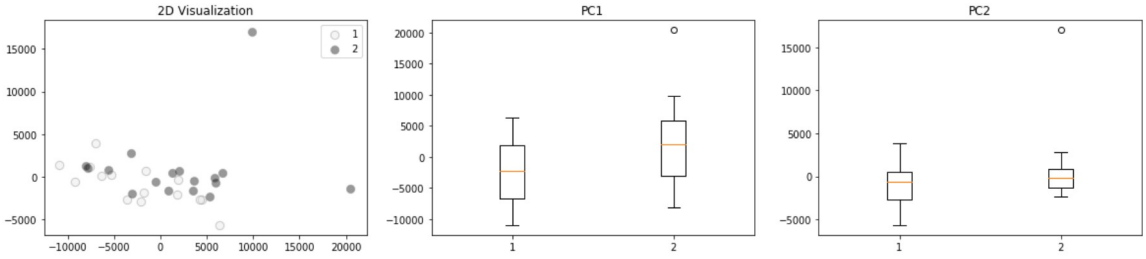

```
===== k = 0.1 , L1 = 0.1  ======
```

```
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 685.1144116934224, tolerance:
504.20847236725206
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 676.217336259386, tolerance: 6
36.8756065497557
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1234.824955222815, tolerance:
682.6001763826207
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 568.9776100222489, tolerance:
564.8443634706733
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2487.825777561981, tolerance:
1168.2610343346375
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 664.2504767014262, tolerance:
433.00811377090014
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1112.6569450264474, tolerance:
934.8281148181023
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 3758.735523306589, tolerance:
2424.9688982482103
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2094.97137461006, tolerance: 1
637.9793170312712
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2562.312307470257, tolerance:
1926.7834575865627
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 3811.982269997713, tolerance:
1446.8529534519632
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2242.861691395376, tolerance:
1480.2810270315763
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2480.4866273775333, tolerance:
1220.2029818292021
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 3491.6832710325198, tolerance:
1761.9056237072994
```

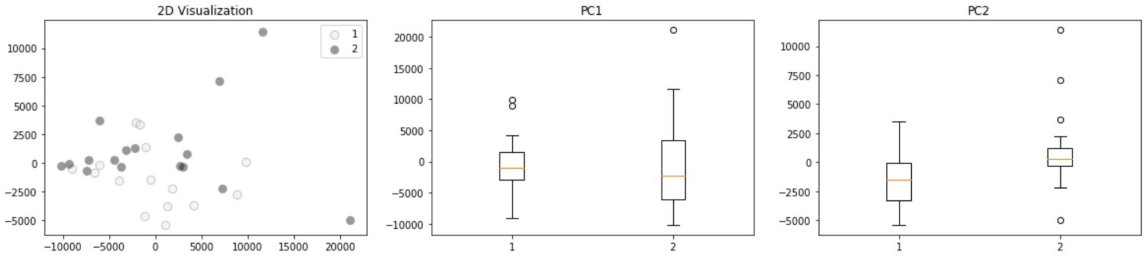
```
f=0.7941234480973275,p=0.3799478592173795
f=0.8744356668595947,p=0.3571968565216328
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
============================================================

------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.9508 2.0000 29.0000  0.7498 0.4814
         Pillai's trace 0.0492 2.0000 29.0000  0.7498 0.4814
 Hotelling-Lawley trace 0.0517 2.0000 29.0000  0.7498 0.4814
    Roy's greatest root 0.0517 2.0000 29.0000  0.7498 0.4814
------------------------------------------------------------

------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.9459 2.0000 29.0000  0.8295 0.4464
         Pillai's trace 0.0541 2.0000 29.0000  0.8295 0.4464
 Hotelling-Lawley trace 0.0572 2.0000 29.0000  0.8295 0.4464
    Roy's greatest root 0.0572 2.0000 29.0000  0.8295 0.4464
============================================================

SVC ACC:  0.7638888888888888
```



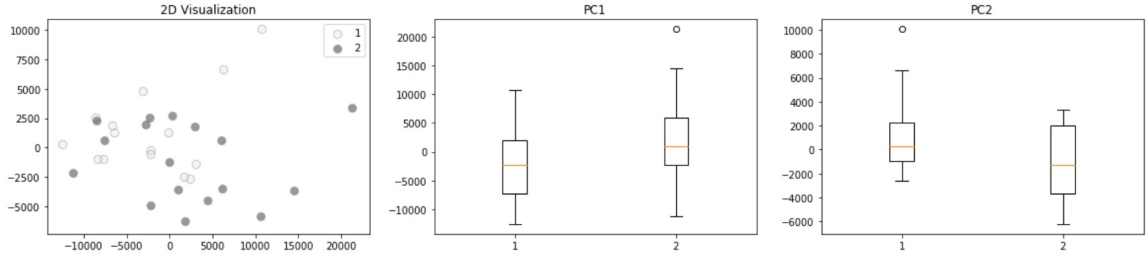

```
===== k = 0.1 , L1 = 1  ======
f=4.2966575063948715,p=0.04687407181404722
f=1.8430429403875888,p=0.18471983237259754
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
============================================================

------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.8315 2.0000 29.0000  2.9391 0.0688
         Pillai's trace 0.1685 2.0000 29.0000  2.9391 0.0688
 Hotelling-Lawley trace 0.2027 2.0000 29.0000  2.9391 0.0688
    Roy's greatest root 0.2027 2.0000 29.0000  2.9391 0.0688
------------------------------------------------------------

------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.8168 2.0000 29.0000  3.2513 0.0532
         Pillai's trace 0.1832 2.0000 29.0000  3.2513 0.0532
 Hotelling-Lawley trace 0.2242 2.0000 29.0000  3.2513 0.0532
    Roy's greatest root 0.2242 2.0000 29.0000  3.2513 0.0532
============================================================

SVC ACC:  0.7941468253968255
```




```
===== k = 0.1 , L1 = 10  ======
f=0.058248913655258854,p=0.8109273359030167
f=4.326647860749346,p=0.046161233069582044
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.8829 2.0000 29.0000  1.9238 0.1642
         Pillai's trace 0.1171 2.0000 29.0000  1.9238 0.1642
 Hotelling-Lawley trace 0.1327 2.0000 29.0000  1.9238 0.1642
    Roy's greatest root 0.1327 2.0000 29.0000  1.9238 0.1642
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.8720 2.0000 29.0000  2.1281 0.1373
         Pillai's trace 0.1280 2.0000 29.0000  2.1281 0.1373
 Hotelling-Lawley trace 0.1468 2.0000 29.0000  2.1281 0.1373
    Roy's greatest root 0.1468 2.0000 29.0000  2.1281 0.1373
================================================================

SVC ACC:  0.7271825396825398
```



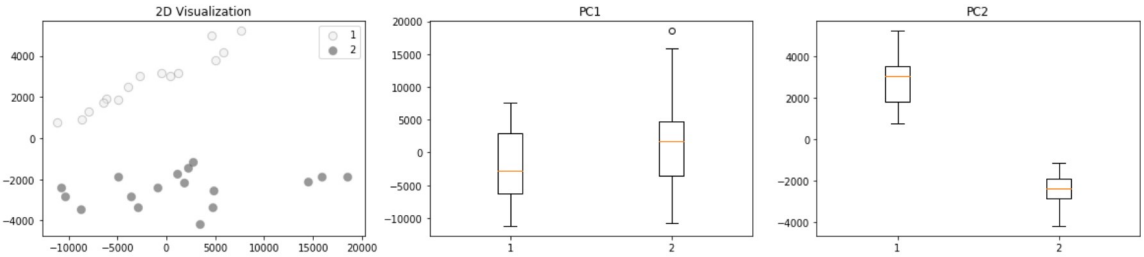
```
===== k = 0.2 , L1 = 0.001  ======
f=2.658059154679605,p=0.11348254760962356
f=4.140535101357306,p=0.05078834064352125
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
============================================================

------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.8132 2.0000 29.0000  3.3318 0.0498
         Pillai's trace 0.1868 2.0000 29.0000  3.3318 0.0498
 Hotelling-Lawley trace 0.2298 2.0000 29.0000  3.3318 0.0498
    Roy's greatest root 0.2298 2.0000 29.0000  3.3318 0.0498
------------------------------------------------------------

------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.7973 2.0000 29.0000  3.6857 0.0375
         Pillai's trace 0.2027 2.0000 29.0000  3.6857 0.0375
 Hotelling-Lawley trace 0.2542 2.0000 29.0000  3.6857 0.0375
    Roy's greatest root 0.2542 2.0000 29.0000  3.6857 0.0375
============================================================

SVC ACC:  0.9722222222222222
```



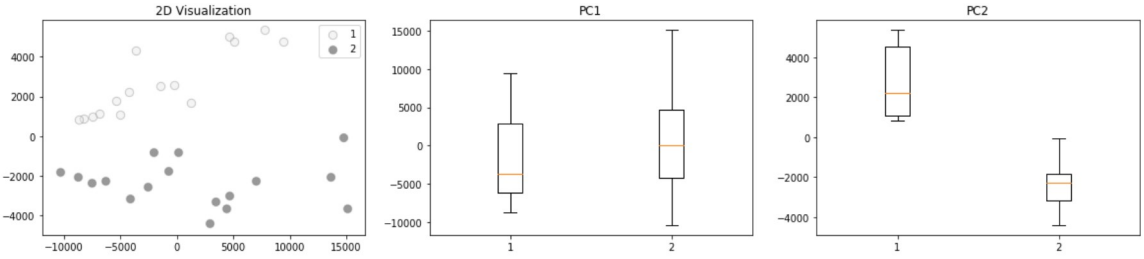

```
===== k = 0.2 , L1 = 0.01  ======
```

```
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1648.3999843725676, tolerance:
1098.170854314675
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 2221.891225126621, tolerance:
1961.2839020810795
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1659.7152402734075, tolerance:
1220.9177422381856
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 1453.6731511108164, tolerance:
1102.733587477971
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 5574.127337507545, tolerance:
4879.619859794887
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 3895.795615467818, tolerance:
3526.0940140691937
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 5609.0871545335085, tolerance:
5242.488328194646
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 6244.175570581737, tolerance:
5044.524647913703
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 12678.75547105898, tolerance:
10325.228789760262
  positive)
C:\Users\eleve\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_des
cent.py:475: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations. Duality gap: 5568.1748168475115, tolerance:
4482.609374454982
  positive)
```

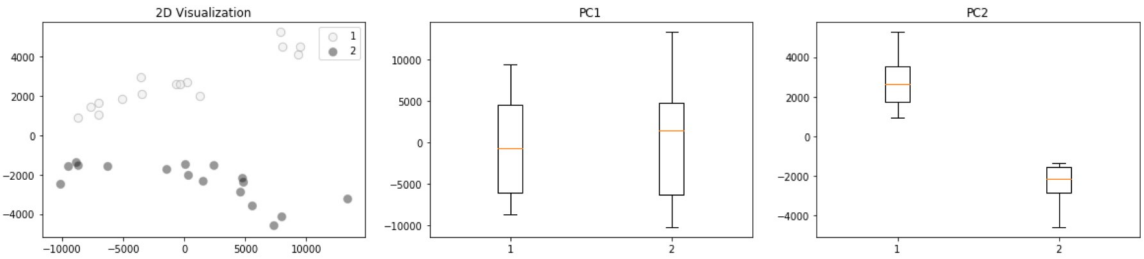
```
f=1.724291902049198,p=0.19909735266364614
f=175.21170496720998,p=4.629902605206098e-14
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1006 2.0000 29.0000 129.6202 0.0000
         Pillai's trace 0.8994 2.0000 29.0000 129.6202 0.0000
 Hotelling-Lawley trace 8.9393 2.0000 29.0000 129.6202 0.0000
    Roy's greatest root 8.9393 2.0000 29.0000 129.6202 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.0918 2.0000 29.0000 143.3866 0.0000
         Pillai's trace 0.9082 2.0000 29.0000 143.3866 0.0000
 Hotelling-Lawley trace 9.8887 2.0000 29.0000 143.3866 0.0000
    Roy's greatest root 9.8887 2.0000 29.0000 143.3866 0.0000
================================================================

SVC ACC:  1.0
```



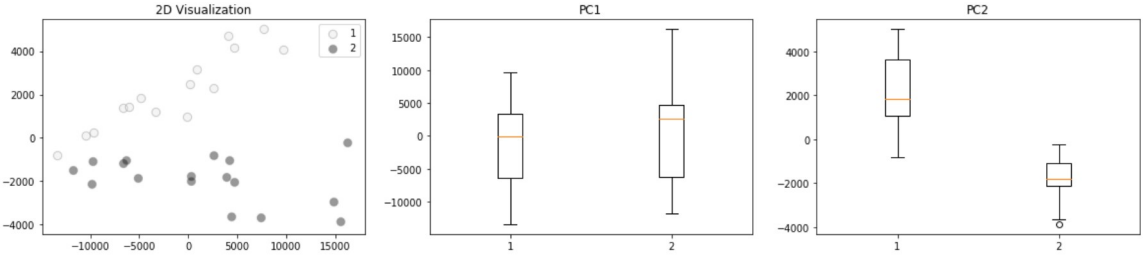

```
===== k = 0.2 , L1 = 0.1  ======
f=1.336008530334896,p=0.25686171105808797
f=97.89328168238266,p=5.85923666548548e-11
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.2081 2.0000 29.0000 55.1852 0.0000
         Pillai's trace 0.7919 2.0000 29.0000 55.1852 0.0000
 Hotelling-Lawley trace 3.8059 2.0000 29.0000 55.1852 0.0000
    Roy's greatest root 3.8059 2.0000 29.0000 55.1852 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1919 2.0000 29.0000 61.0462 0.0000
         Pillai's trace 0.8081 2.0000 29.0000 61.0462 0.0000
 Hotelling-Lawley trace 4.2101 2.0000 29.0000 61.0462 0.0000
    Roy's greatest root 4.2101 2.0000 29.0000 61.0462 0.0000
================================================================

SVC ACC:  0.9444444444444444
```



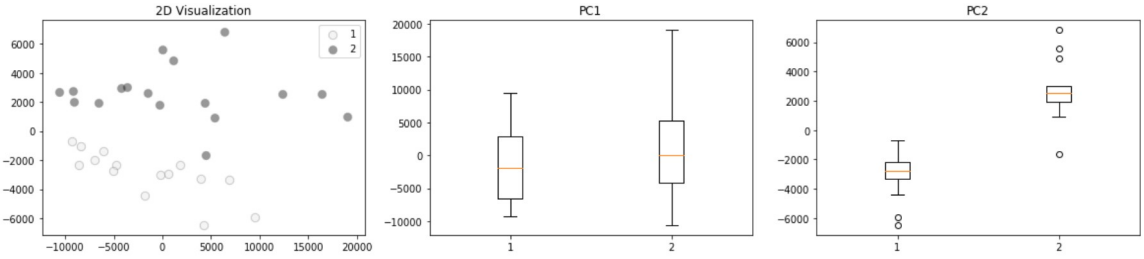

```
===== k = 0.2 , L1 = 1  ======
f=0.14773745500814037,p=0.7034181285033814
f=149.62170429158795,p=3.4530041297328493e-13
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1763 2.0000 29.0000 67.7464 0.0000
         Pillai's trace 0.8237 2.0000 29.0000 67.7464 0.0000
 Hotelling-Lawley trace 4.6722 2.0000 29.0000 67.7464 0.0000
    Roy's greatest root 4.6722 2.0000 29.0000 67.7464 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1621 2.0000 29.0000 74.9414 0.0000
         Pillai's trace 0.8379 2.0000 29.0000 74.9414 0.0000
 Hotelling-Lawley trace 5.1684 2.0000 29.0000 74.9414 0.0000
    Roy's greatest root 5.1684 2.0000 29.0000 74.9414 0.0000
================================================================

SVC ACC:  0.9166666666666666
```



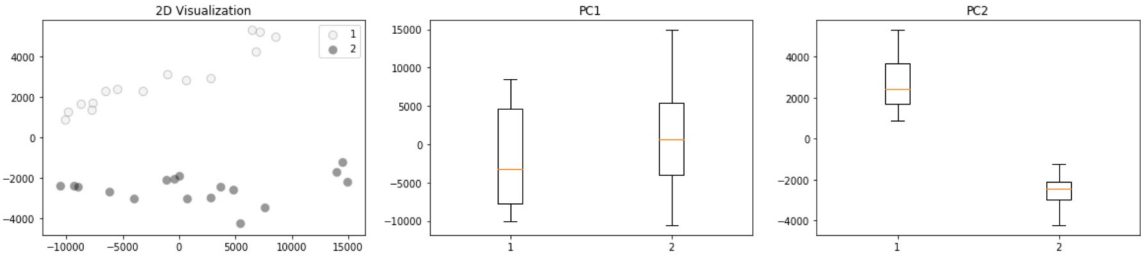
```
===== k = 0.2 , L1 = 10  ======
f=1.199788484435583,p=0.28208002935523246
f=63.75091540887328,p=6.523947170906379e-09
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.3024 2.0000 29.0000 33.4496 0.0000
         Pillai's trace 0.6976 2.0000 29.0000 33.4496 0.0000
 Hotelling-Lawley trace 2.3069 2.0000 29.0000 33.4496 0.0000
    Roy's greatest root 2.3069 2.0000 29.0000 33.4496 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.2815 2.0000 29.0000 37.0021 0.0000
         Pillai's trace 0.7185 2.0000 29.0000 37.0021 0.0000
 Hotelling-Lawley trace 2.5519 2.0000 29.0000 37.0021 0.0000
    Roy's greatest root 2.5519 2.0000 29.0000 37.0021 0.0000
================================================================

SVC ACC:  0.8854166666666666
```



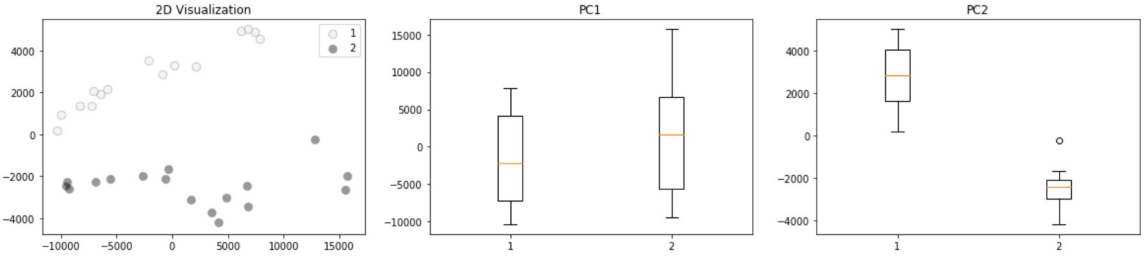

```
===== k = 0.3 , L1 = 0.001  ======
f=1.285999471719434,p=0.2657673068121966
f=78.41433845648044,p=7.170831586806164e-10
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.2543 2.0000 29.0000 42.5255 0.0000
         Pillai's trace 0.7457 2.0000 29.0000 42.5255 0.0000
 Hotelling-Lawley trace 2.9328 2.0000 29.0000 42.5255 0.0000
    Roy's greatest root 2.9328 2.0000 29.0000 42.5255 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.2356 2.0000 29.0000 47.0420 0.0000
         Pillai's trace 0.7644 2.0000 29.0000 47.0420 0.0000
 Hotelling-Lawley trace 3.2443 2.0000 29.0000 47.0420 0.0000
    Roy's greatest root 3.2443 2.0000 29.0000 47.0420 0.0000
================================================================

SVC ACC:  1.0
```



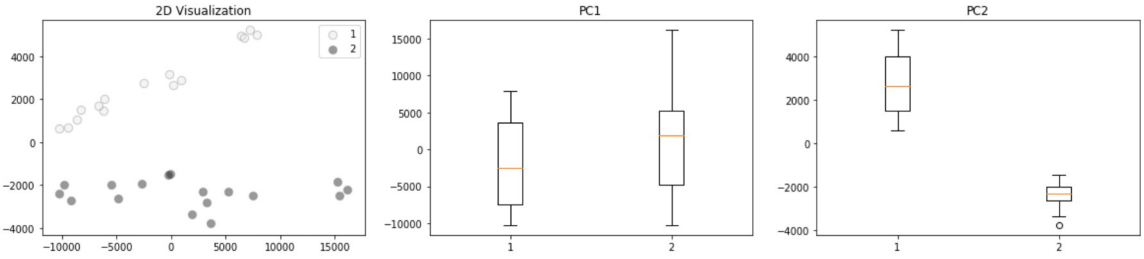

```
===== k = 0.3 , L1 = 0.01  ======
f=1.7212688966600238,p=0.1994808310215053
f=179.98582061057704,p=3.273186457707603e-14
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
        Intercept        Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.0971 2.0000 29.0000 134.8290 0.0000
         Pillai's trace 0.9029 2.0000 29.0000 134.8290 0.0000
 Hotelling-Lawley trace 9.2985 2.0000 29.0000 134.8290 0.0000
    Roy's greatest root 9.2985 2.0000 29.0000 134.8290 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y             Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda  0.0886 2.0000 29.0000 149.1486 0.0000
         Pillai's trace  0.9114 2.0000 29.0000 149.1486 0.0000
 Hotelling-Lawley trace 10.2861 2.0000 29.0000 149.1486 0.0000
    Roy's greatest root 10.2861 2.0000 29.0000 149.1486 0.0000
================================================================

SVC ACC:  1.0
```



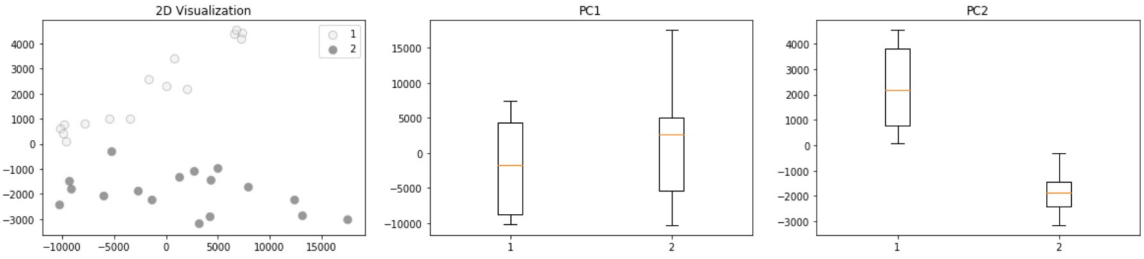
```
===== k = 0.3 , L1 = 0.1  ======
f=1.6747152781910628,p=0.20550221035108424
f=143.45245234577348,p=5.852117863984197e-13
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
        Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1312 2.0000 29.0000 96.0464 0.0000
         Pillai's trace 0.8688 2.0000 29.0000 96.0464 0.0000
 Hotelling-Lawley trace 6.6239 2.0000 29.0000 96.0464 0.0000
    Roy's greatest root 6.6239 2.0000 29.0000 96.0464 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y             Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1201 2.0000 29.0000 106.2470 0.0000
         Pillai's trace 0.8799 2.0000 29.0000 106.2470 0.0000
 Hotelling-Lawley trace 7.3274 2.0000 29.0000 106.2470 0.0000
    Roy's greatest root 7.3274 2.0000 29.0000 106.2470 0.0000
================================================================

SVC ACC:  1.0
```



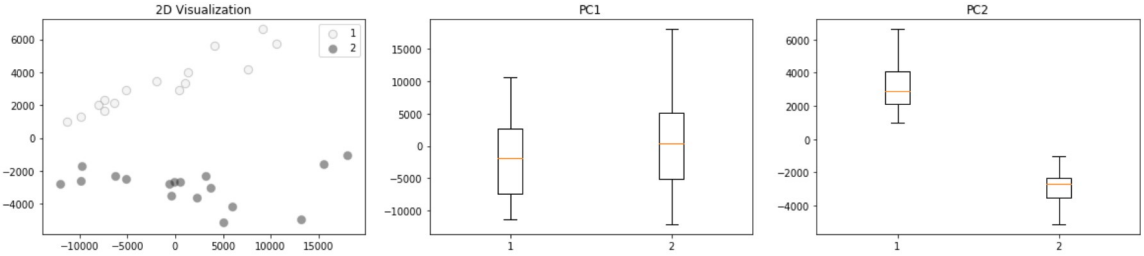

```
===== k = 0.3 , L1 = 1  ======
f=1.7804638023672994,p=0.19213287259429498
f=143.099357247646,p=6.0349511069829e-13
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
        Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1281 2.0000 29.0000 98.6510 0.0000
         Pillai's trace 0.8719 2.0000 29.0000 98.6510 0.0000
 Hotelling-Lawley trace 6.8035 2.0000 29.0000 98.6510 0.0000
    Roy's greatest root 6.8035 2.0000 29.0000 98.6510 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y             Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1173 2.0000 29.0000 109.1283 0.0000
         Pillai's trace 0.8827 2.0000 29.0000 109.1283 0.0000
 Hotelling-Lawley trace 7.5261 2.0000 29.0000 109.1283 0.0000
    Roy's greatest root 7.5261 2.0000 29.0000 109.1283 0.0000
================================================================

SVC ACC:  0.9444444444444444
```



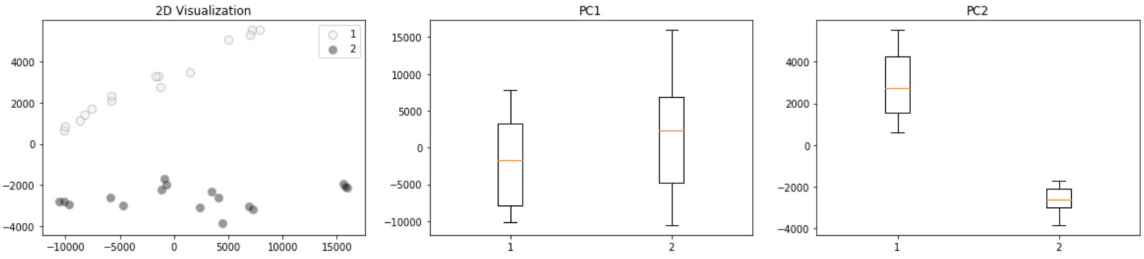

```
===== k = 0.3 , L1 = 10  ======
f=1.6129959507164962,p=0.21383388079884294
f=84.18839188733098,p=3.263762913987765e-10
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.2290 2.0000 29.0000 48.8092 0.0000
         Pillai's trace 0.7710 2.0000 29.0000 48.8092 0.0000
 Hotelling-Lawley trace 3.3661 2.0000 29.0000 48.8092 0.0000
    Roy's greatest root 3.3661 2.0000 29.0000 48.8092 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.2117 2.0000 29.0000 53.9930 0.0000
         Pillai's trace 0.7883 2.0000 29.0000 53.9930 0.0000
 Hotelling-Lawley trace 3.7237 2.0000 29.0000 53.9930 0.0000
    Roy's greatest root 3.7237 2.0000 29.0000 53.9930 0.0000
================================================================

SVC ACC:  0.9444444444444444
```



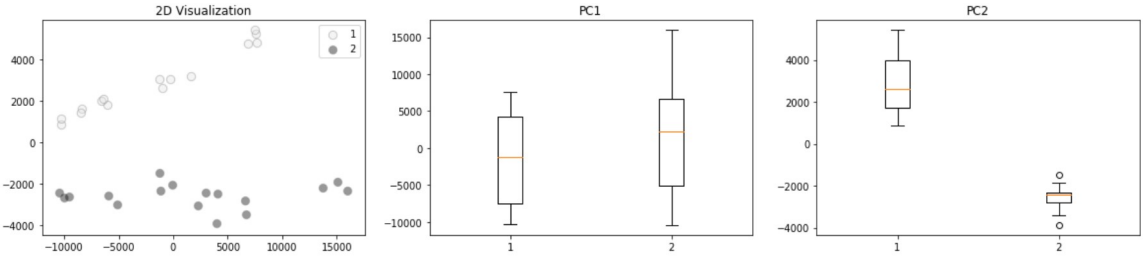
```
===== k = 0.4 , L1 = 0.001  ======
f=1.068507537376773,p=0.3095441681046772
f=154.54479015571607,p=2.2960341803484834e-13
['PC1', 'PC2']
['Intercept', 'y']
                 Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1399 2.0000 29.0000 89.1614 0.0000
         Pillai's trace 0.8601 2.0000 29.0000 89.1614 0.0000
 Hotelling-Lawley trace 6.1491 2.0000 29.0000 89.1614 0.0000
    Roy's greatest root 6.1491 2.0000 29.0000 89.1614 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1282 2.0000 29.0000 98.6308 0.0000
         Pillai's trace 0.8718 2.0000 29.0000 98.6308 0.0000
 Hotelling-Lawley trace 6.8021 2.0000 29.0000 98.6308 0.0000
    Roy's greatest root 6.8021 2.0000 29.0000 98.6308 0.0000
================================================================

SVC ACC:  0.9722222222222222
```



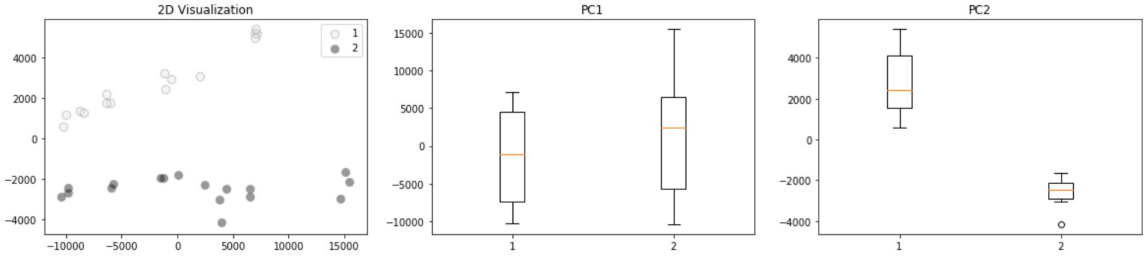

```
===== k = 0.4 , L1 = 0.01  ======
f=2.135114539818177,p=0.1543555513544617
f=157.54097638143037,p=1.8006900476903468e-13
['PC1', 'PC2']
['Intercept', 'y']
                   Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1024 2.0000 29.0000 127.0484 0.0000
         Pillai's trace 0.8976 2.0000 29.0000 127.0484 0.0000
 Hotelling-Lawley trace 8.7620 2.0000 29.0000 127.0484 0.0000
    Roy's greatest root 8.7620 2.0000 29.0000 127.0484 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.0935 2.0000 29.0000 140.5417 0.0000
         Pillai's trace 0.9065 2.0000 29.0000 140.5417 0.0000
 Hotelling-Lawley trace 9.6925 2.0000 29.0000 140.5417 0.0000
    Roy's greatest root 9.6925 2.0000 29.0000 140.5417 0.0000
================================================================

SVC ACC:  1.0
```



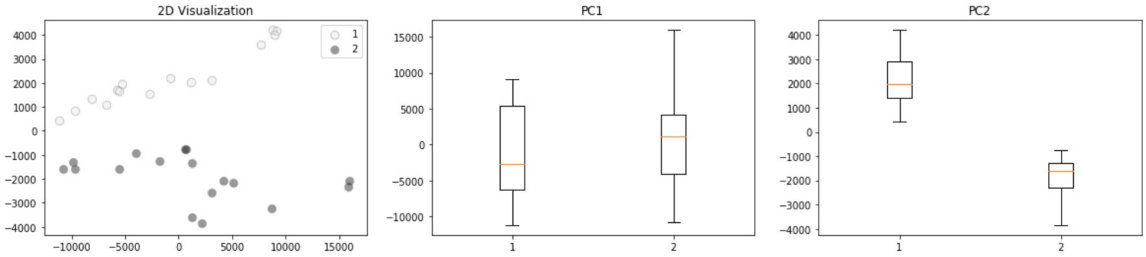

```
===== k = 0.4 , L1 = 0.1  ======
f=1.6376009151142177,p=0.21046333946071988
f=184.66081838039918,p=2.3485385856009768e-14
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
        Intercept        Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.0964 2.0000 29.0000 135.8551 0.0000
         Pillai's trace 0.9036 2.0000 29.0000 135.8551 0.0000
 Hotelling-Lawley trace 9.3693 2.0000 29.0000 135.8551 0.0000
    Roy's greatest root 9.3693 2.0000 29.0000 135.8551 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y             Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda  0.0880 2.0000 29.0000 150.2837 0.0000
         Pillai's trace  0.9120 2.0000 29.0000 150.2837 0.0000
 Hotelling-Lawley trace 10.3644 2.0000 29.0000 150.2837 0.0000
    Roy's greatest root 10.3644 2.0000 29.0000 150.2837 0.0000
================================================================

SVC ACC:  1.0
```



```
===== k = 0.4 , L1 = 1  ======
f=1.747318565456052,p=0.196205730734803
f=156.90838627664965,p=1.8948707466927791e-13
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1154 2.0000 29.0000 111.1749 0.0000
         Pillai's trace 0.8846 2.0000 29.0000 111.1749 0.0000
 Hotelling-Lawley trace 7.6672 2.0000 29.0000 111.1749 0.0000
    Roy's greatest root 7.6672 2.0000 29.0000 111.1749 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1055 2.0000 29.0000 122.9823 0.0000
         Pillai's trace 0.8945 2.0000 29.0000 122.9823 0.0000
 Hotelling-Lawley trace 8.4815 2.0000 29.0000 122.9823 0.0000
    Roy's greatest root 8.4815 2.0000 29.0000 122.9823 0.0000
================================================================

SVC ACC:  1.0
```




```
===== k = 0.4 , L1 = 10  ======
f=0.6382368857090238,p=0.43062974045971825
f=115.33656099825497,p=8.46632373721209e-12
['PC1', 'PC2']
['Intercept', 'y']
                  Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept        Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.2013 2.0000 29.0000 57.5217 0.0000
         Pillai's trace 0.7987 2.0000 29.0000 57.5217 0.0000
 Hotelling-Lawley trace 3.9670 2.0000 29.0000 57.5217 0.0000
    Roy's greatest root 3.9670 2.0000 29.0000 57.5217 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
           y            Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
          Wilks' lambda 0.1856 2.0000 29.0000 63.6309 0.0000
         Pillai's trace 0.8144 2.0000 29.0000 63.6309 0.0000
 Hotelling-Lawley trace 4.3883 2.0000 29.0000 63.6309 0.0000
    Roy's greatest root 4.3883 2.0000 29.0000 63.6309 0.0000
================================================================

SVC ACC:  1.0
```



NOTE:
No matter we use the sparse representation z in latent space, or the reconstructed data in the origial space, the 2D visualizations after PCA are the same. Therefore, only the visualization of reconstructed x is shown.

When k = 0.1, the original discriminative structures in the data is lost (some samples from two classes begin to overlap), and the MANOVA under some λ test cannot reject the null-hypothesis H_0.
k = 0.2 well preserves the discriminative structure. This result also supports the four-to-one rule.

In [34]:
```
# Show K-fold ACC and MANOVA p-value in a table

from IPython.core.display import display, HTML

s = '<table>'

for key in result:
    s+= '<tr>'

    s+= '<td>' + key[:7] + '</td>'
    s+= '<td>' + key[7:] + '</td>'

    s+= '<td>' + str( round(result[key][0],3) ) + '</td>'
    s+= '<td>' + result[key][1].replace("Wilks' lambda             ","")[:12] +
'</td>'

    s+= '</tr>'

s += '</table>'
display(HTML(s))
```

k = 0.1	L1 = 0.001	0.783	0.0775615 Pi
k = 0.1	L1 = 0.01	0.881	0.000135201
k = 0.1	L1 = 0.1	0.764	0.446361 Pil
k = 0.1	L1 = 1	0.794	0.0532109 Pi
k = 0.1	L1 = 10	0.727	0.137285 Pil
k = 0.2	L1 = 0.001	0.972	0.0374761 Pi
k = 0.2	L1 = 0.01	1.0	9.2009e-16 P
k = 0.2	L1 = 0.1	0.944	4.03419e-11
k = 0.2	L1 = 1	0.917	3.48757e-12
k = 0.2	L1 = 10	0.885	1.04321e-08
k = 0.3	L1 = 0.001	1.0	7.88588e-10
k = 0.3	L1 = 0.01	1.0	5.47152e-16
k = 0.3	L1 = 0.1	1.0	4.49393e-14
k = 0.3	L1 = 1	0.944	3.19245e-14
k = 0.3	L1 = 10	0.944	1.67097e-10
k = 0.4	L1 = 0.001	0.972	1.15588e-13
k = 0.4	L1 = 0.01	1.0	1.19766e-15
k = 0.4	L1 = 0.1	1.0	4.9497e-16 P
k = 0.4	L1 = 1	1.0	6.84319e-15
k = 0.4	L1 = 10	1.0	2.47689e-11