Αρχικά δημιουργούμε τις συναρτήσεις που ορίσαμε στα 2 προηγούμενα ερωτήματα και θα τις χρειαστούμε για την επεξεργασία των σημάτων σε αυτή την άσκηση.

In [1]:

```
import scipy.signal
def square(list):
    return [i ** 2 for i in list] #square every element in a list

def ste(x,fs): #Compute short-time energy.
    win=np.hamming(20*fs) #Declaration of a hamming window of length 20 sec
    return scipy.signal.convolve(square(np.abs(x)), win, mode="same")#Returns convolution with the hamming window
of length 20s
```

In [2]:

```
import numpy as np
def features (signal): #Computation of mean , min ,max & std value of a signal
    mean=np.mean(signal)
    minimum=np.min(signal)
    maximum=np.max(signal)
    std=np.std(signal)
    return (mean,minimum,maximum,std) #returns a tuple with the characteristics
```

In [3]:

```
def teo_help (signal,n): #Help function that computes TEO operator at points
    if (n==np.size(signal)-1):  #at the end signal(n+1)=signal(n)
        return (((signal[n])**2)-signal[n-1]*signal[n])
    if (n==0): #at start signal(n-1)=signal(n)
        return (((signal[0])**2)-signal[0]*signal[1])
    return ((signal[n])**2 - signal[n-1]*signal[n+1]) #if we are not at the start or at the end that returns TEO
at a mid-point

def teo (signal): #Teager Energy Operator
    y=[]
    for i in range (0,len(signal)): #Use teo_help len(signal) times
        y.append(teo_help(signal,i))
    return y
```

In [4]:

```
import scipy as sp
def Gabor_help (n,fc,a,fs): #help function that computes Gabor fun at points
    b=a/fs
    Omega_c=(2*(np.pi))*(fc/fs)
    b_sq=b**2
    return np.exp(-b_sq*(n**2))*np.cos(Omega_c*n)

def Gabor (fc,a,fs): #Help function that computes Gabor h(n) function
    b=a/fs
    N=int ((3/b)+1)
    y=[]
    for i in range(-N,N+1):
        y.append(Gabor_help(i,fc,a,fs))
    return y

def gaborfilt(shma,fc,a,fs): #function that filters a signal with Gabor filter
    return scipy.signal.lfilter(Gabor(fc,a,fs),1,shma)
```

In [5]:

```
def filtering (shma,fs): #Functions that filters a signal with 25 Gabor filter at different frequencies
    K=25
    a=fs/(2*K)
    fcmin=0 #starting frequency
    fcmax=(fs/2) #final frequency
    d=fcmax-fcmin # subspace between starting and final frequency
    sub_of_filts=d/25 #subspace between two consecutive frequencies
    ans=[] #list of the 25 filtered signals
    for j in range(0,25):
        y=[]
        fc_pr=fcmin+(j*sub_of_filts) #Central frequency in filtering the signal
        y.append(gaborfilt(shma,fc_pr,a,fs))
        ans.append(list(y))
    return ans #return 25 lists => 25 times Gabor filtered signal at different frequencies
```

In [6]:

```
def binomial (shma):
    return scipy.signal.lfilter ([.25, .5 ,.25],1,shma)
```

In [7]:

```
def hamming_windowing (shma,fs):
    win=np.hamming(fs*20) #hamming window of length 20 sec
    shift=fs*5  # Shift between two consecutive windows
    lim=np.size(shma)-np.size(win)+1 #max value where we can window the signal
    ap=[] #return list with the winowed signals
    for i in range (0,lim,shift):
           ap.append(shma[i:i+len(win)]*win)
    return ap   #return the windows of the signal
```

In [8]:

```
def Mean_Teager_Energy(shma): #function that computes mean value of a list only for positive values
    return np.mean(list(filter(lambda num: num > 0, shma)))
```

Φορτώνουμε τα σήματα step & sleep μέσω της εντολής np.load(). Σε αρκετά κελιά κάνω στο τέλος *print('finished')* ώστε να ξέρω πότε τελείωσε η εκτελεσή τους (καθώς κάποια αργούσαν).

In [9]:

```
step='step_' #δημιουργώ strings με τα ονόματα των μεταβλητών ώστε να τα φορτώσω όλα με ένα loop
sleep='sleep_'
npz='.npz'
#Acc in step signals
step_list_accx=[]
step_list_accy=[]
step_list_accz=[]
#Gyroscope in step signals
step_list_gyrx=[]
step_list_gyry=[]
step_list_gyrz=[]
#Hrm in step signals
step_list_hrm=[]
#Acc in sleep signals
sleep_list_accx=[]
sleep_list_accy=[]
sleep_list_accz=[]
#Gyroscope in sleep signals
sleep_list_gyrx=[]
sleep_list_gyry=[]
sleep_list_gyrz=[]
#Hrm in sleep signals
sleep_list_hrm=[]
#Store 20 steps signals
for i in range(0,20):
    accx=[]
    accy=[]
    accz=[]
    gyrx=[]
    gyry=[]
    gyrz=[]
    if (i<10):
        i_str='0'+str(i) #αυτό το κάνω διότι τα σήματα από 1 έως 9 γράφονται ως step_0X.npz
    else :
        i_str = str(i)
    data = np.load(step+i_str+npz)
    for x in data['acc'] :
        accx.append(x[0]) # append has O(1) complexity so it's ok
        accy.append(x[1])
        accz.append(x[2])
    step_list_accx.append(accx)
    step_list_accy.append(accy)
    step_list_accz.append(accz)
    for x in data['gyr'] :
        gyrx.append(x[0])
        gyry.append(x[1])
        gyrz.append(x[2])
    step_list_gyrx.append(gyrx)
    step_list_gyry.append(gyry)
    step_list_gyrz.append(gyrz)
    step_list_hrm.append(data['hrm'])
print('finished')
```

```
finished
```

Κάνω pad με μηδενικά ώστε να στρογγυλέψω το μέγεθος του πίνακα και να μην βγω εκτός ορίων στην παραθύρωση με παράθυρο Hamming .

In [10]:

```
def pad_zeros (shma:list,num): #function that pads zeros to a list
    for i in range (num):
        shma.append(0)
    return shma #Returns zero-padded signal

for i in range (0,20):  #προσθέτω μηδενικά στα σήματα ώστε να έχουν πιο στρογγυλό μήκος
    pad_zeros(step_list_accx[i],(12000-np.size(step_list_accx[i])))
    pad_zeros(step_list_accy[i],(12000-np.size(step_list_accy[i])))
    pad_zeros(step_list_accz[i],(12000-np.size(step_list_accz[i])))

    pad_zeros(step_list_gyrx[i],(12000-np.size(step_list_gyrx[i])))
    pad_zeros(step_list_gyry[i],(12000-np.size(step_list_gyry[i])))
    pad_zeros(step_list_gyrz[i],(12000-np.size(step_list_gyrz[i])))
    step_list_hrm[i]=list(step_list_hrm[i])
    pad_zeros(step_list_hrm[i],(3000-np.size(step_list_hrm[i])))
print('finished')
```

```
finished
```

In [11]:

```
#Store 20 sleep signals
for i in range(0,20):
    accx=[]
    accy=[]
    accz=[]
    gyrx=[]
    gyry=[]
    gyrz=[]
    if (i<10):
        i_str='0'+str(i)
    else :
        i_str = str(i)
    data = np.load(sleep+i_str+npz)
    for x in data['acc'] :
        accx.append(x[0])
        accy.append(x[1])
        accz.append(x[2])
    sleep_list_accx.append(accx)
    sleep_list_accy.append(accy)
    sleep_list_accz.append(accz)
    for x in data['gyr'] :
        gyrx.append(x[0])
        gyry.append(x[1])
        gyrz.append(x[2])
    sleep_list_gyrx.append(gyrx)
    sleep_list_gyry.append(gyry)
    sleep_list_gyrz.append(gyrz)
    sleep_list_hrm.append(data['hrm'])
print('finished')
```

```
finished
```

In [12]:

```
for i in range (0,20): #pad με μηδενικά ώστε να στρογγυλοποιήσω το μέγεθος του πίνακα
    pad_zeros(sleep_list_accx[i],(12000-np.size(sleep_list_accx[i])))
    pad_zeros(sleep_list_accy[i],(12000-np.size(sleep_list_accy[i])))
    pad_zeros(sleep_list_accz[i],(12000-np.size(sleep_list_accz[i])))

    pad_zeros(sleep_list_gyrx[i],(12000-np.size(sleep_list_gyrx[i])))
    pad_zeros(sleep_list_gyry[i],(12000-np.size(sleep_list_gyry[i])))
    pad_zeros(sleep_list_gyrz[i],(12000-np.size(sleep_list_gyrz[i])))
    sleep_list_hrm[i]=list(step_list_hrm[i])
    pad_zeros(sleep_list_hrm[i],(3000-np.size(sleep_list_hrm[i])))
print('finished')
```

```
finished
```

In [13]:

```
def Mean_Teager (shma,fs):
    ap=[]#λίστα όπου θα αποθηκεύσουμε την μέση τιμή από κάθε Teager Energy
    a=hamming_windowing(shma,fs) #Στο a θα αποθηκευτούν τα 116 (μετά από δοκιμές παρατήρησα ότι τόσα βγαίνουν) παράθυρα
    A=np.shape(a)[0] #Στο A θα αποθηκευτεί ο #παραθύρων (δηλαδή 116)
    filtra=[] #εδώ θα αποθηκευτούν τα 116 παράθυρα ώστε το καθένα να φιλτραριστεί 25 φορές
    for i in range(0,A): #φιλτράρω το κάθε παράθυρο , αυτό θα μου επιστρέψει λίστα(#παραθύρων X 25 φιλτραρισμένες λίστες)
        filtra.append(filtering(a[i],fs))
    for i in range (0,A): #Εφαρμόζω Teager τελεστή ενέργειας σε κάθενα από τα 25 παράθυρα του σήματος X όλα τα παράθυρα (
σσ.116)
        for j in range (0,25):
            filtra[i][j][0]=teo(filtra[i][j][0])
    for i in range(0,A):
        for j in range (0,25):#Παιρνώ κάθε Teager Energy από binomial φίλτρο 2 φορές
            filtra[i][j][0]=binomial(filtra[i][j][0])
            filtra[i][j][0]=binomial(filtra[i][j][0])
    mesh_Teager_list=[] #λίστα όπου αποθηκεύω την μέση τιμή μη-μηδενικών τιμών Μέσης-Teager Ενέργειας
    max_mean_list=[]
    for i in range (0,A):
        mesh_Teager_list=[] #λίστα όπου αποθηκεύω την μέση τιμή μη-μηδενικών τιμών Μέσης-Teager Ενέργειας
        for j in range (0,25):
            mesh_Teager_list.append(Mean_Teager_Energy(filtra[i][j][0]))
        max_mean_list.append(mesh_Teager_list)
    ap=[]
    for i in max_mean_list:
        ap.append(np.max(i))
    return ap
```

In [15]:

```
X=[] #40 X 56 Matrix
for i in range(0,20):
    print(i)
    a=[]
    #now we append STE features
    helper=ste(step_list_accx[i],20)
    f=features(helper) #features return a tuple with (mean,min,max,std)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(step_list_accy[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(step_list_accz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(step_list_gyrx[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(step_list_gyry[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(step_list_gyrz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(step_list_hrm[i],5)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    #now we append Max Teager features
    helper=Mean_Teager(step_list_accx[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
```

```
    a.append(f[2])

    a.append(f[3])
    helper=Mean_Teager(step_list_accy[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(step_list_accz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(step_list_gyrx[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(step_list_gyry[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(step_list_gyrz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(step_list_hrm[i],5)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    X.append(a)#αποθηκεύω το διάνυσμα 1 X 56 με τα χαρακτηριστικά των ενεργειών των σημάτων στον πίνακα X
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

In [16]:

```
for i in range(0,20):
    print(i)
    a=[]
    #now we append STE features
    helper=ste(sleep_list_accx[i],20)
    f=features(helper) #features return a tuple with (mean,min,max,std)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(sleep_list_accy[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(sleep_list_accz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
```

```
    a.append(f[2])

    a.append(f[3])
    helper=ste(sleep_list_gyrx[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(sleep_list_gyry[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(sleep_list_gyrz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=ste(sleep_list_hrm[i],5)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    #now we append Max Teager features
    helper=Mean_Teager(sleep_list_accx[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(sleep_list_accy[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(sleep_list_accz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(sleep_list_gyrx[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(sleep_list_gyry[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(sleep_list_gyrz[i],20)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    helper=Mean_Teager(sleep_list_hrm[i],5)
    f=features(helper)
    a.append(f[0])
    a.append(f[1])
    a.append(f[2])
    a.append(f[3])
    X.append(a) #αποθηκεύω το διάνυσμα 1 X 56 με τα χαρακτηριστικά των ενεργειών των σημάτων στον πίνακα X
print('finished')
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
finished
```

In [17]:

```
import pandas as pd
#τυπώνω τον πίνακα X με την βοήθεια της βιβλιοθήκης pandas
pinakas = pd.DataFrame(X)
pinakas
```

Out[17]:

	0	1	2	3	4	5	6	7	8	
0	439.166576	2.501818	5249.778010	817.353790	410.127491	3.545461	2881.288973	507.375817	392.232779	4.4006
1	938.307927	0.716901	4752.606603	952.195537	1037.332812	1.531718	3595.106241	985.202074	847.288367	1.5852
2	1058.716647	21.843070	6936.209669	1026.451313	935.306994	41.646780	4025.390565	746.204494	749.781612	37.6112
3	1127.012317	66.804522	6092.795128	972.675866	769.686526	128.445237	2915.052780	574.110249	636.730077	3.5394
4	504.437748	0.213636	3375.750515	658.587364	587.605838	2.308482	4203.701750	683.387174	386.200057	0.1897
5	1208.226767	3.487672	4252.132681	987.529001	1124.417137	7.131201	3852.656060	921.496035	1109.296157	5.9184
6	408.654238	0.089442	2394.352907	394.829772	523.970951	0.597309	3515.292906	594.280684	448.082039	0.2202
7	1684.182353	278.870857	5918.308183	947.606456	1735.318956	219.967360	6511.649847	1043.471433	1350.456271	162.5653
8	1086.777897	1.097953	5047.269352	1236.166498	875.840231	0.874386	3715.994537	910.208290	773.271283	0.5972
9	565.005032	0.460705	4587.222876	749.140736	552.768426	1.728895	3214.216658	642.715546	590.822650	1.2486
10	2946.518796	18.009614	10017.046072	1733.597917	2984.811410	10.249714	17164.222242	2291.269813	2285.719482	4.7563
11	673.323597	1.034640	2214.316616	470.330803	598.152582	6.843798	2659.516978	497.384291	542.338872	2.9955
12	1952.271412	12.464836	5993.310335	1251.637227	1939.556787	14.712403	6664.081006	1463.552590	1802.361572	0.3529
13	1908.853230	103.720380	6963.489245	1345.133813	1772.428754	101.119889	6624.321632	1203.941533	1673.666116	45.3330
14	117.325471	1.053018	931.350649	174.570079	186.240139	2.891288	669.752320	222.584822	155.606526	1.7969
15	258.452576	7.750661	2287.668849	358.613012	216.464099	11.717251	1588.879960	303.845053	155.785632	5.4945
16	1249.534894	10.899836	5281.286733	1095.614982	1354.216617	31.924491	6942.309851	1202.375936	804.307948	27.2410
17	560.429078	0.764083	3239.457725	591.995958	819.017551	1.515014	4567.122361	769.563445	523.265188	2.6990
18	1588.854016	6.926776	6837.275792	1475.208682	1235.070745	11.077322	10153.872480	1320.402702	1153.764061	11.7773
19	2477.888082	0.609925	10677.067657	2253.184327	2240.672500	0.670113	11971.210041	2562.762239	1580.128910	0.6656
20	2.127751	0.061298	90.889651	10.307951	0.834874	0.061011	36.644117	4.149663	1.565898	0.1569
21	4.248800	1.061181	114.822334	12.841864	0.870714	0.148772	23.092249	2.628069	6.040962	0.8000
22	4.891707	2.395876	5.267335	0.225591	2.517314	1.273617	3.138054	0.147173	2.920211	1.2385
23	3.885705	0.449445	47.567908	5.907960	1.275428	0.160267	14.391634	2.454418	4.049265	0.1291
24	0.518175	0.127445	9.400358	1.193403	0.545343	0.103443	9.234601	1.219217	0.299263	0.0746
25	2.545421	1.124337	3.001401	0.202026	1.060474	0.435274	2.606292	0.227737	2.849918	1.4457
26	55.909026	0.043285	1094.538281	143.300033	76.869971	0.093092	1952.616589	240.483901	72.101571	0.0553
27	0.223396	0.099252	0.538167	0.039954	1.258532	0.629831	1.632125	0.092683	1.020642	0.4426
28	0.206628	0.057075	4.857269	0.540543	0.479456	0.181917	6.650512	0.718355	1.175095	0.3855
29	0.089237	0.031774	0.115055	0.008922	0.177121	0.081014	0.252685	0.022585	0.178925	0.0656
30	8.235802	0.156718	405.683292	47.173067	38.985697	0.215338	2095.664860	240.049735	22.067480	0.5032
31	25.770076	0.198123	1339.397619	154.025666	40.098497	0.144792	2119.361988	243.567601	24.093154	0.3388
32	8.134938	0.862774	247.975884	28.500630	6.441451	0.182905	305.987396	36.031238	14.861773	1.8268
33	59.117211	0.095411	1216.409447	187.484213	54.050474	0.328362	1000.041016	171.657817	64.700213	0.1786
34	6.577612	0.199651	299.054039	34.822241	7.101071	0.593811	242.001455	28.999955	7.372541	0.2835
35	3.149166	0.584820	29.917935	3.339728	1.662189	0.260227	50.630816	5.832327	3.937582	0.8952
36	47.593584	0.044198	1433.452777	200.771852	73.200047	0.053300	2240.632114	312.116745	89.954684	0.1872
37	42.375481	0.064650	1453.422020	179.024168	42.232337	0.092747	1154.970670	157.584850	36.056624	0.0643
38	3.616300	1.168582	4.887594	0.433040	0.346533	0.162061	0.581485	0.066121	3.533883	1.5911
39	59.867104	0.171297	2649.165893	317.953712	82.015920	0.216057	3728.620061	457.531035	43.802643	0.1932

40 rows × 56 columns

In [18]:

```
M=X-np.mean(X,axis=0)#κανονικοποιούμε τον πίνακα ώστε να έχει μηδενική μέση τιμή
M_svd=np.linalg.svd(M) #Κάνουμε Singular Value Decomposition , το αποτέλεσμα αυτής της πράξης θα μας επιστρέψει 3 πίνακες
```

In [19]:

```
U=M_svd[0] #U ο πίνακας με τα ιδιοδιανύσματα του X*X^T
S=M_svd[1] #Σ το διάνυσμα με τις μη μηδενικές ιδιοτιμές του X*X^T
Vh=M_svd[2] #V ο πίνακας με τα ιδιοδιανύσματα του X^T*X
```

In [20]:

```
L_k=[]#αποθηκεύω τις τιμές σι^2/40 όπου σi οι ιδιοτιμές του X*X^T ώστε να βρω αργότερα τα αντίστοιχα ποσοστά
for x in M_svd[1]:#που θα μου δώσουν τις πρωτεύουσες και δευτερεύουσες συνιστώσες
    L_k.append((x**2)/40)
L_k
```

Out[20]:

```
[225327256729388.75,
 30623978227949.543,
 19099752676513.43,
 4598372372548.694,
 1858553886194.3906,
 1044918146297.8406,
 246066146033.0887,
 76929194721.77711,
 64719019494.80647,
 11407322337.797209,
 6526182839.618216,
 3646879560.862279,
 2249444976.831018,
 1716751227.495027,
 886319148.6305956,
 502511756.0975356,
 137323964.80204052,
 73537883.064466,
 30098626.632338416,
 14662381.14801979,
 4608281.852535504,
 2293495.93665299,
 699120.8574574066,
 264142.4167062546,
 111971.20034950358,
 70033.80361007825,
 43146.01150828728,
 30764.81735279,
 12860.006778843604,
 5313.845851048742,
 1549.4123809967275,
 849.2339455732081,
 506.74701333887424,
 115.25488720980657,
 38.14754263693989,
 11.934426613078566,
 9.164583525422104,
 0.6943632485280117,
 0.13604891168122712,
 1.6258797076593816e-18]
```

In [21]:

```
V=Vh.transpose() #Αντιστροφή του πίνακα V^T (χρησιμοποιώ T αντί για H καθώς είναι πραγματικός πίνακας)
```

In [22]:

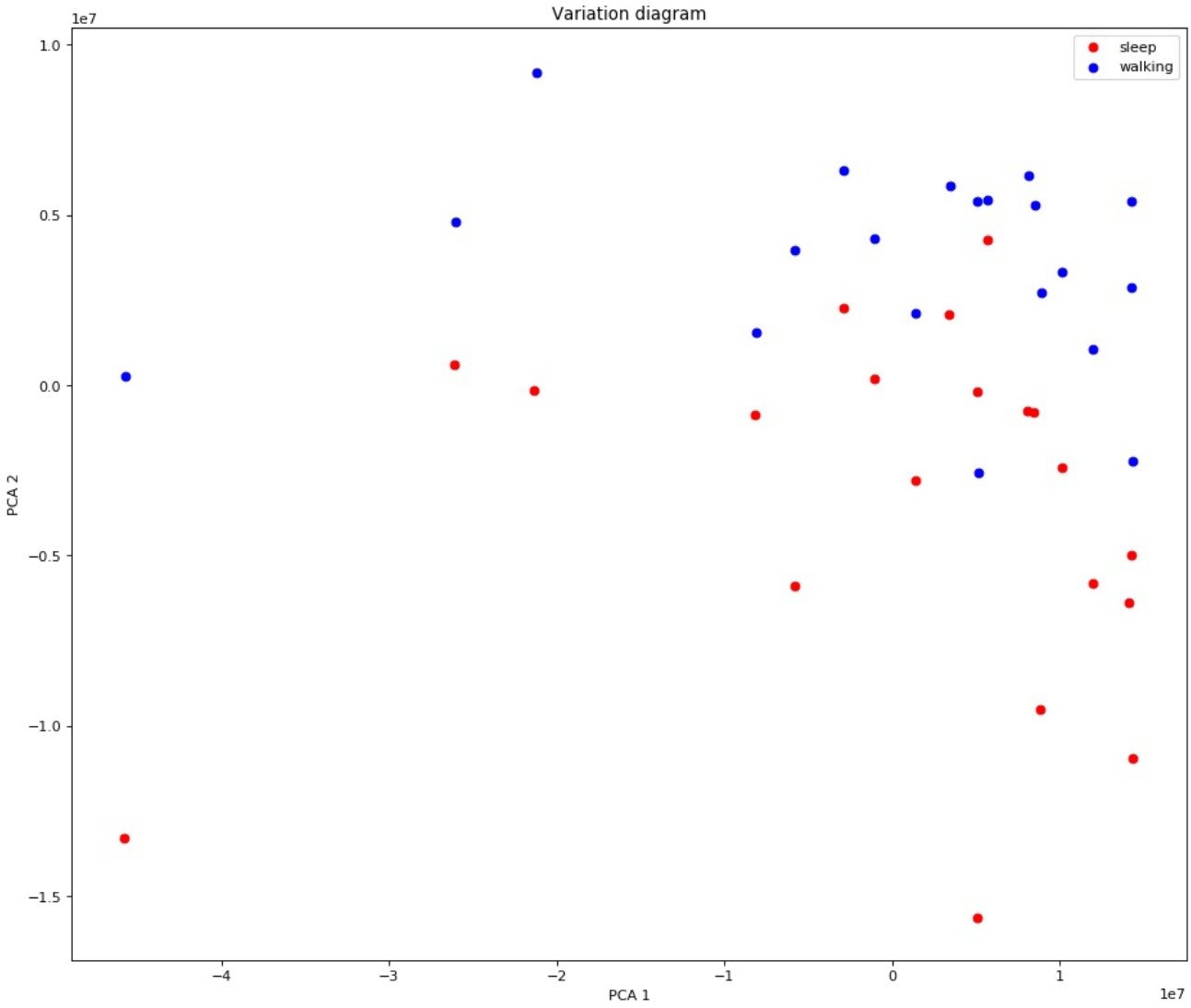
```
sum_lk=0 # Υπολογισμός του αθροίσματος όλων των λκ = σi^2/40 ώστε να βρω το ποσοστό του καθενός μετά
for x in L_k:
    sum_lk=sum_lk+x
print(sum_lk)
L_k_pososta=[]
for x in L_k:
    L_k_pososta.append(x/sum_lk) #Ποσοστό = λκ /sum(λκ για κάθε κ )
L_k_final=sorted(L_k_pososta)
print(L_k_final)
```

```
282967745576056.8
[5.7458128464410914e-33, 4.807930013516665e-16, 2.4538600578466953e-15, 3.2387378663123337e-14, 4.217
5925700587665e-14, 1.3481233544579552e-13, 4.073075076990641e-13, 1.7908295954623898e-12, 3.00116871
5693601e-12, 5.475579479358973e-12, 1.8778980764153816e-11, 4.5446899796524894e-11, 1.08721993350019
3e-10, 1.5247678289429062e-10, 2.4749747879393687e-10, 3.9570305131970476e-10, 9.3347182085548893e-10
, 2.4706733130808l9e-09, 8.105149694654997e-09, 1.6285537573033666e-08, 5.1816439778924547e-08, 1.06
36769420862643e-07, 2.598807963598816e-07, 4.852990029746349e-07, 1.7758623163022981e-06, 3.13222677
3147784e-06, 6.066950224309555e-06, 7.949474850045785e-06, 1.2887969098520668e-05, 2.306334535171992
e-05, 4.031315411788203e-05, 0.00022871518223058793, 0.0002718655957242303, 0.0008695907921666302, 0
.003692711139817823, 0.006568076804693076, 0.01625051775136941, 0.06749798510650305, 0.1082242718710
0853, 0.7963001446354765]
```

Παρατηρώ πως για p=1 έχουμε το 79,64% της πληροφορίας . Για p=2 το 90% και για p=3 το 96% . Άρα για p=3 έχουμε πάνω από το 95% της πληροφορίας του σήματος .Για μεγαλύτερα p οι προσθήκες είναι αμελητέες.

In [24]:

```
K=[]
for i in range (56):
    y_k=np.matmul(M,V[:,i]) #Πολλαπλασιασμός του πίνακα M με τον πίνακα V
    K.append(y_k) #yk=M * Vk

x1=K[0][0:20] #σήματα ύπνου
x2=K[0][20:40] #σήματα περπατήματος
y1=K[1][0:20] #σήματα ύπνου
y2=K[1][20:40] #σήματα περπατήματος

import matplotlib.pyplot as plt

fig=plt.figure(figsize=(14, 12), dpi= 80)

plt.scatter(x1,y1,color='red')
plt.scatter(x2,y2,color='blue')

plt.xlabel('PCA 1')
plt.ylabel('PCA 2')

plt.title('Variation diagram')
plt.legend(["sleep","walking"])

plt.show()
```



α) Παρατηρούμε ότι το 78% της πληροφορίας περιέχεται στην πρωτεύουσα συνιστώσα . Το 90% περιέχουν οι 2 πρώτες πρωτεύουσες συνιστώσες και το 97% οι τρεις πρώτες πρωτεύουσες συνιστώσες . Από έκει και πέρα η αύξηση του ποσοστού είναι αμελητaία . Αυτή η προσέγγιση λέγεται Low Rank Approximation . Προσεγγιστικά αυτό συμβαίνει γιατί είναι λογικό όλα τα διανύσματα των δεδομένων να μην είναι ανεξάρτητα μεταξύ τους . Παρόμοια ιδέα έχουμε δει σε μαθήματα ελέγχου όπου ένα σύστημα προσεγγίζεται μόνο από τις μεγάλες ιδιοτιμές (επικρατούντες πόλοι) και εν τέλει απομένουν λιγότερες γραμμές σε έναν πίνακα που περιγράφει το σύστημα .

β) Το παραπάνω scatter plot απεικονίζει τις δύο πρωτεύουσες συνιστώσες . Οι κόκκινες κουκίδες αντιστοιχούν σε βήματα ύπνου και οι μπλε σε σήματα βημάτων . Το παραπάνω scatter plot δεν μας ικανοποιεί τόσο η εικόνα που εμφανίζει αλλά οι κουκίδες που αντιστοιχούν σε βήματα φαίνονται απεικονίζονται ψηλότερα από τις κουκίδες που αντιστοιχούν σε σήματα ύπνου.

Υλοποίση του αλγορίθμου K-means.

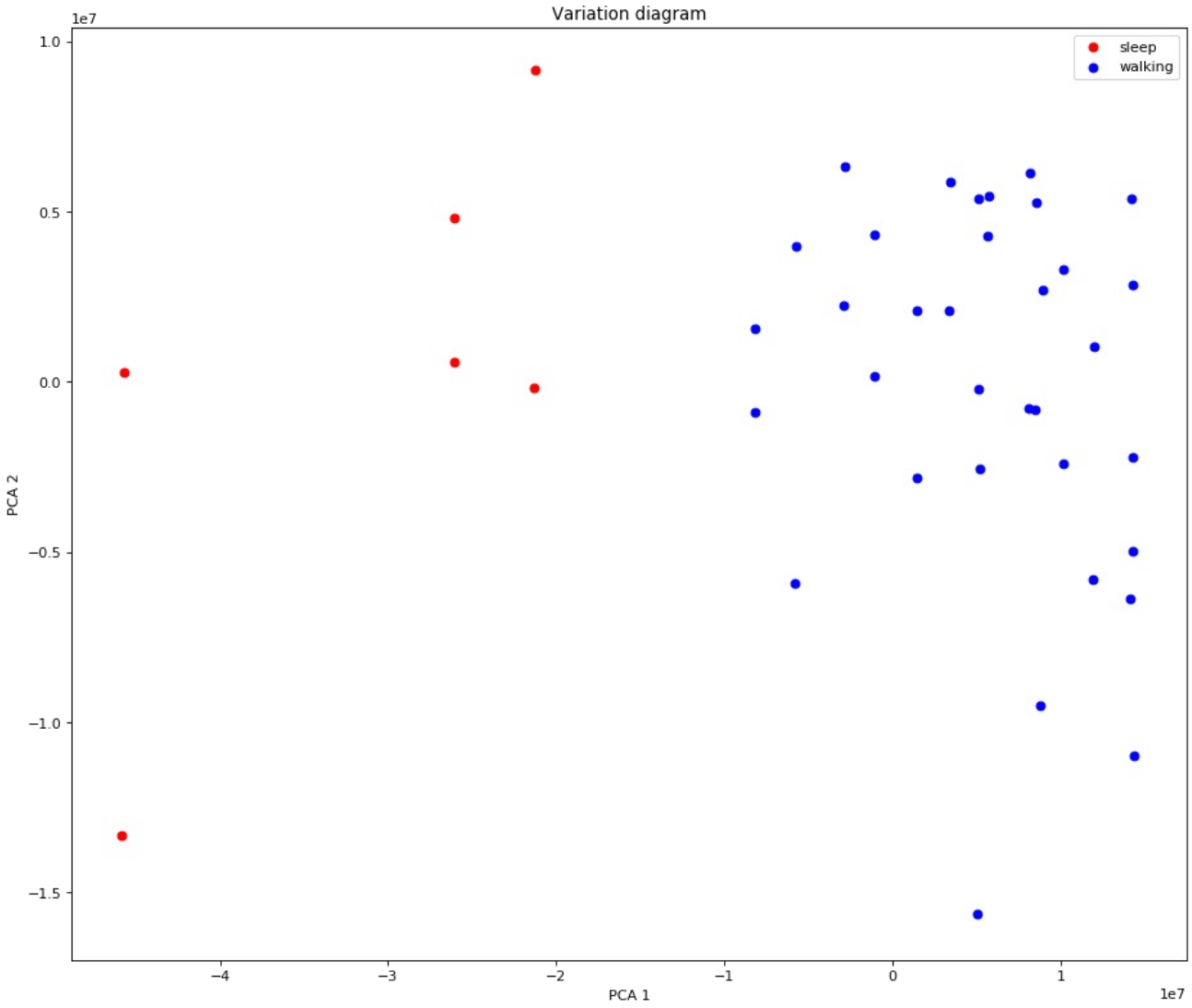
In [25]:

```
import random

Mat=np.array(K[0:2][:])
T=Mat.transpose() #αντιστροφή των 2 πρώτων γραμμών του πίνακα K που περιείχε τις 2 πρώτες πρωτεύουσες συνιστώσες

curD1=1 #αρχικοποίηση της τρέχουσας απόστασης από το κέντρο center1
curD2=1 #αρχικοποίηση της τρέχουσας απόστασης από το κέντρο center2

center_1=T[int(random.uniform(0, 40-1))]# τυχαία αρχικοποίηση των δύο κέντρων
center_2=T[int(random.uniform(0, 40-1))]#B είναι ο πίνακας 40 X 2 που περιέχει τις 2 πρωτεύουσες συνιστώσες

Similarity_matrix=np.ones(40) #φτιάχνω έναν πίνακα όπου θα σημειώνω με 0 για sleep και 1 για step ώστε να συγκρίνω με τα πριν
Prev=np.zeros(40) # πίνακας όπου θα αποθηκεύσω τα σήματα ύπνου και περπατήματος πριν τον αλγόριθμο K-means ώστε να τα συγκρίνω έπειτα
for i in range (20,40):
    Prev[i]=1 #1 πριν για step

def closer_to (x,y,z):
    x_y=np.linalg.norm(x-y) # L2 νόρμα του (X - Y)
    x_z=np.linalg.norm(x-z) #L2 νόρμα του (X - Z)
    if (x_y < x_z) :
        return True #Το y είναι πιο κοντά στο x απότι στο z
    else :
        return False #Το z είναι πιο κοντά στο x απότι στο y

def paramorfwsh (past_D1,past_D2,center1,center2):  # συνάρτηση που μετρά την παραμόρφωση των κέντρων ώστε να αποφανθώ μετά από ένα κατόφλι
    sum1=0 #την διακοπή του while loop
    sum2=0 #τα δύο αθροίσματα των αποστάσεων από τα κέντρα
    for i in range (0,40):
        sum1 += np.linalg.norm(T[i]-center1)
        sum2 += np.linalg.norm(T[i]-center2)
    if (sum1/curD1 <= 1.05 and sum2/curD2 <= 1.05): #Αν το άθροισμα των αποστάσεων δεν έχει αλλαγές άνω του 5% σε σχέση με πριν τότε βγαίνω από το loop
        return (True,sum1,sum2)
    else :
        return (False,sum1,sum2)

L=2
#print(len(T))
while (42):
    Cluster_1=[] #αποθηκεύω τις κουκίδες που έχουν κέντρο το center_1 εδώ
    Cluster_2=[] #αποθηκεύω τις κουκίδες που έχουν κέντρο το center_2 εδώ
    for i in range (0,40):
        if (closer_to(T[i],center_1,center_2)):
            Cluster_1.append(T[i]) #εάν είναι πιο κοντά στο center_1 τότε θα τοποθετηθεί στο cluster_1
            Similarity_matrix[i]=0 #διότι μπορεί σε προηγούμενη επανάληψη να ήταν 0
        else :
            Cluster_2.append(T[i]) #εάν είναι πιο κοντά στο center_2 τότε θα τοποθετηθεί στο cluster_2
            Similarity_matrix[i]=1 #αλλάζω και την τιμή στον πίνακα για την σύγκριση αργότερα

    center_1=[np.mean(Cluster_1[:][0]),np.mean(Cluster_1[:][1])] #επανυπολογισμός του κέντρου 1 ως το μέσο όλων των σημείων σε X και Y
    center_2=[np.mean(Cluster_2[:][0]),np.mean(Cluster_2[:][1])]  #επανυπολογισμός του κέντρου 2 ως το μέσο όλων των σημείων σε X και Y
    rouf=paramorfwsh(curD1,curD2,center_1,center_2) #ελέγχω κατά πόσο άλλαξε το άθρισμα όλων των αποστάσεων σημαντικά σε σχέση με πριν
    if (rouf[0]==True):
        break #εάν δεν άλλαξε βγαίνω από την λούπα
    else :
        curD1=rouf[1] #αλλιώς ενημερώνω τα current sum of distances ώστε να τα συγκρίνω με τα επόμενα
        curD2=rouf[2]

print('finished')
```

```
finished
```

In [26]:

```
C_1= np.array(Cluster_1) #εδώ περιέχονται τα σημεία περπατήματος
C_2= np.array(Cluster_2) #εδώ περιέχονται τα σημεία ύπνου

fig=plt.figure(figsize=(14, 12), dpi= 80)

plt.scatter(C_1[:,0],C_1[:,1],color='red')
plt.scatter(C_2[:,0],C_2[:,1],color='blue')
plt.title('Variation diagram')
plt.legend(['sleep','walking'])
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')

plt.show()
```



Δείκτης Rand

In [27]:

```
from sklearn.metrics.cluster import adjusted_rand_score
print(adjusted_rand_score(Similarity_matrix,Prev))
```

```
-0.013250194855806838
```

Αυτός ο δείκτης μας επιβεβαιώνει την αρχική μας υπόθεση πως πρέπει να έγινε κάποιο λάθος στο προηγούμενο διάγραμμα . Για αυτό άλλωστε έχει μικρή τιμή . Τρέχοντας αρκετές φορές τον αλγόριθμο K-means παρατηρώ πως αλλάζουν σε ένα μικρό βαθμό τα αποτελέσματα που τυπώνει .

Τέλος εργαστηριακής άσκησης 3