Assignment07

May 16, 2019

Assignment07
Assignment07: Apply K-means algorithm to both image value and its spatial domain
Software Engineering
20154652 Lee Dong Jae

In [1]:
```
import PIL.Image as pilimg
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Read image
im = pilimg.open('xavi.jpg')
```

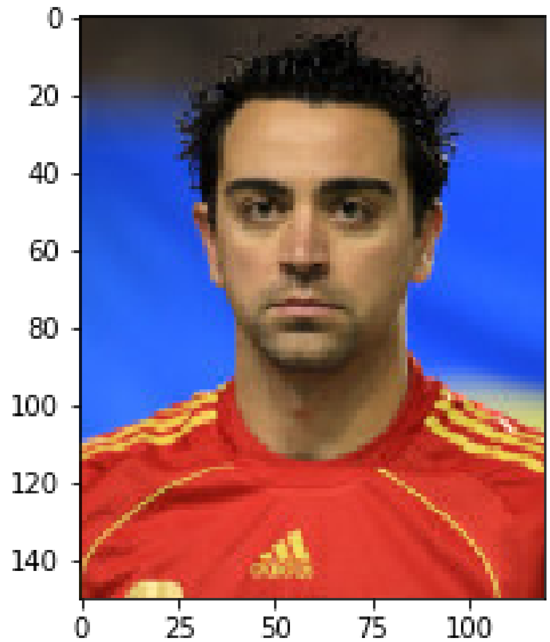
In [3]:
```
#change the image to numpy array
data = np.array(im)
```

In [4]:
```
print(data.shape)
```

```
(150, 120, 3)
```

In [5]:
```
plt.imshow(data)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x210cb1bcac8>
```



In [6]:
```
#make a r,g,b value
r = data[:,:,0].astype(float)
g = data[:,:,1].astype(float)
b = data[:,:,2].astype(float)
#get mean of r,g,b
r_sum = 0
g_sum = 0
b_sum = 0
for i in range(len(r)):
    for j in range(len(r[0])):
        r_sum += r[i][j]
        g_sum += g[i][j]
        b_sum += b[i][j]

r_mean = r_sum / (len(r[0])*len(r))
g_mean = g_sum / (len(r[0])*len(r))
b_mean = b_sum / (len(r[0])*len(r))
#get std of r,g,b
r_diff = 0
g_diff = 0
b_diff = 0
for i in range(len(r)):
    for j in range(len(r[0])):
        r_diff += (r_mean - r[i][j])**2
        g_diff += (g_mean - g[i][j])**2
        b_diff += (b_mean - b[i][j])**2
```

```
        r_std = (r_diff / (len(r[0])*len(r)))**0.5
        g_std = (g_diff / (len(r[0])*len(r)))**0.5
        b_std = (b_diff / (len(r[0])*len(r)))**0.5
```

```
In [7]: hor = list(data.shape)[0]
        ver = list(data.shape)[1]
```

```
In [8]: def distance(x, y):
            d = (x - y) ** 2
            #s = np.sum(d)
            #r = np.sqrt(s)

            return(d)
```

```
In [9]: #make a matrix that contain value of row
        matrix_row = [ [col for row in range(ver)] for col in range(hor)]
        matrix_row = np.array(matrix_row)

        #make a matrix that contain value of col
        matrix_col = [ [row for row in range(ver)] for col in range(hor)]
        matrix_col = np.array(matrix_col)
```

```
In [10]: #scale the domain data
         def minmax_scaler(matrix):
             minmax_matrix = (matrix - np.min(matrix))/(np.max(matrix) - np.min(matrix))
             return minmax_matrix

         matrix_row = minmax_scaler(matrix_row)
         matrix_col = minmax_scaler(matrix_col.T)
         matrix_col = matrix_col.T

         #scale the r,g,b data
         def standard_scaler(matrix):
             standard_matrix = (matrix - np.mean(matrix)) / (np.std(matrix))
             return standard_matrix

         r = standard_scaler(r)
         g = standard_scaler(g)
         b = standard_scaler(b)
```

```
In [11]: def initialize_label(k, alpha):

             #Assign a dictionary that contain energy
             info = {'energy_list': []}

             #initialize label randomly
             first_label = np.random.randint(k, size = (hor, ver))
             clusters_hor = {idx: [] for idx in range(k)}
             clusters_ver = {idx: [] for idx in range(k)}
```

```
        new_clusters = {idx: [] for idx in range(k)}

        #append corresponding [hor, ver] of image to cluster
        for i in range(k):
            clusters_hor[i], clusters_ver[i] = np.where(first_label == i)
            for j in range(len(clusters_hor[i])):
                new_clusters[i].append([clusters_hor[i][j], clusters_ver[i][j]])

        make_new_centroid(new_clusters, k, info, alpha)
```

In [12]:
```
def energy_function(centroid, clusters, k, alpha):
    value_energy = 0
    domain_energy = 0
    energy = 0

    #get energy
    for m in range(k):
        for n in clusters[m]:
            value_energy += distance(r[n[0]][n[1]], centroid[m][0])
            value_energy += distance(g[n[0]][n[1]], centroid[m][1])
            value_energy += distance(b[n[0]][n[1]], centroid[m][2])
            domain_energy += distance(matrix_row[n[0]][n[1]], centroid[m][3])
            domain_energy += distance(matrix_col[n[0]][n[1]], centroid[m][4])

    energy = value_energy + (alpha*domain_energy)

    return energy / (hor*ver)
```

In [13]:
```
def make_new_centroid(clusters, k, info, alpha, centroid = 0):
    new_centroid = np.zeros((k,5))

    #sum previous data contained in same cluster
    for i in range(k):
        for j in clusters[i]:
            new_centroid[i][0] += r[j[0]][j[1]]
            new_centroid[i][1] += g[j[0]][j[1]]
            new_centroid[i][2] += b[j[0]][j[1]]
            new_centroid[i][3] += matrix_row[j[0]][j[1]]
            new_centroid[i][4] += matrix_col[j[0]][j[1]]

    for m in range(k):
        if(len(clusters[m])!=0):
            new_centroid[m][0] = new_centroid[m][0] / len(clusters[m])
            new_centroid[m][1] = new_centroid[m][1] / len(clusters[m])
            new_centroid[m][2] = new_centroid[m][2] / len(clusters[m])
            new_centroid[m][3] = new_centroid[m][3] / len(clusters[m])
            new_centroid[m][4] = new_centroid[m][4] / len(clusters[m])
    #if clustering does not change over, plot images and information
```

```
        if np.array_equal(centroid, new_centroid):
            print('end')
            print('\n\n')
            print("K = ", k)
            print("Lambda = ", alpha)
            plot_image(new_centroid, clusters, k, alpha)
            plot_charts(info)
        else:
            do_clustering(new_centroid, k, info, alpha)
```

In [14]:
```
def do_clustering(centroid, k, info, alpha):
    #make a place for put indexes of data to each clusters
    clusters = {idx: [] for idx in range(k)}

    #a temporary array for keeping the distance
    temp_value_distance = np.zeros(k)
    temp_domain_distance = np.zeros(k)
    temp_distance = np.zeros(k)

    for i in range(hor):
        for j in range(ver):
            for t in range(k):
                temp_value_distance[t] += distance(r[i][j], centroid[t][0])
                temp_value_distance[t] += distance(g[i][j], centroid[t][1])
                temp_value_distance[t] += distance(b[i][j], centroid[t][2])
                temp_domain_distance[t] += distance(matrix_row[i][j], centroid[t][3])
                temp_domain_distance[t] += distance(matrix_col[i][j], centroid[t][4])

                temp_distance[t] = temp_value_distance[t] + (alpha*temp_domain_distanc
            #find the argmin of distance. And append a idx of data to the cluster[argm
            temp_min = min(temp_distance)
            min_index = np.where(temp_distance == temp_min)
            clusters[min_index[0][0]].append([i, j])
            temp_distance = np.zeros(k)
            temp_value_distance = np.zeros(k)
            temp_domain_distance = np.zeros(k)

    #calcuate energy at each time
    energy2 = energy_function(centroid, clusters, k, alpha)
    print(energy2)

    #append calculated information to each list

    info['energy_list'].append(energy2)
    make_new_centroid(clusters, k, info, alpha, centroid)
```

In [15]:
```
def plot_charts(info):
    plt.figure(figsize=(10, 8))
```

```
            plt.title("Energy")
            plt.plot(range(len(info['energy_list'])), info['energy_list'])
            plt.show()
```

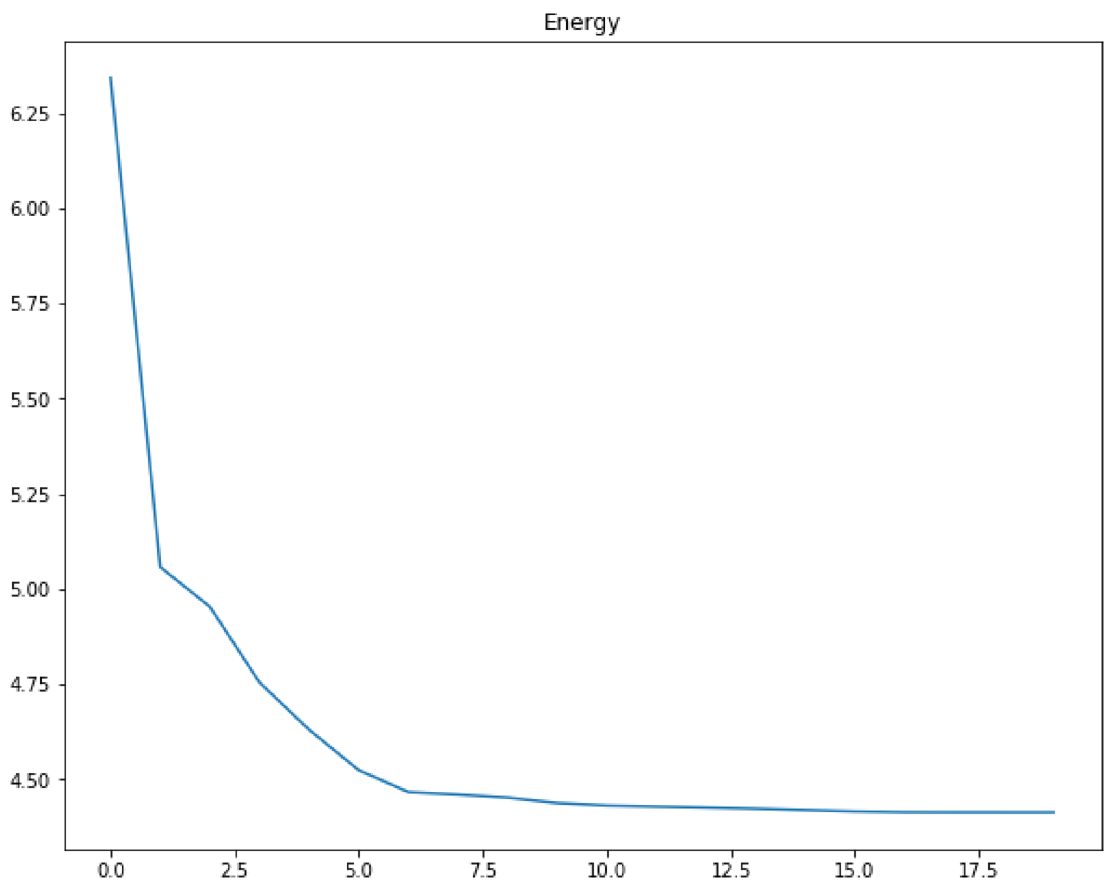
```
In [16]: def plot_image(centroid, clusters, k, alpha):
             plt.figure(1)
             new_image = np.zeros((hor, ver, 3), dtype = np.uint8)
             for i in range(k):
                 for j in clusters[i]:
                     new_image[j[0]][j[1]][0] = ((centroid[i][0]*r_std)+r_mean).astype(int)
                     new_image[j[0]][j[1]][1] = ((centroid[i][1]*g_std)+g_mean).astype(int)
                     new_image[j[0]][j[1]][2] = ((centroid[i][2]*b_std)+b_mean).astype(int)
             plt.title("new image")
             plt.imshow(new_image)
             frame = plt.gca()
             frame.axes.get_xaxis().set_visible(False)
             frame.axes.get_yaxis().set_visible(False)
             plt.show()
```

```
In [17]: initialize_label(2, 20)
```

```
6.342616306681541
5.058046197156479
4.953995456210616
4.755101814041372
4.6309397659673674
4.524666039581248
4.4672680128931725
4.460851691226436
4.452746700095637
4.438426265921549
4.431867442702257
4.429187959903213
4.426567874097767
4.423672911694446
4.419841186114865
4.416235805588511
4.414391299208631
4.414101700647147
4.4140766280622215
4.4140750288065105
end
```

```
K =  2
Lambda =  20
```


new image

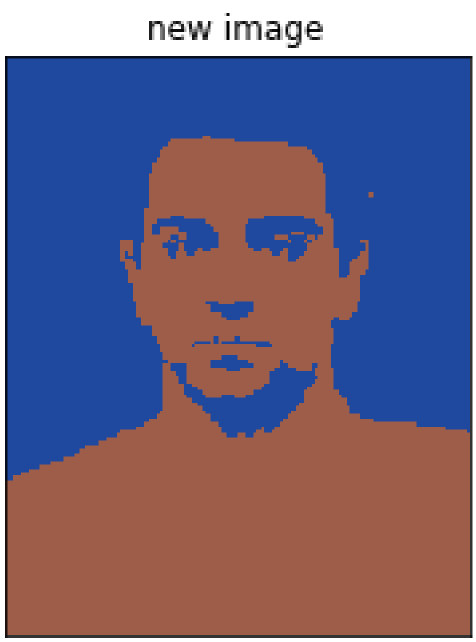

Energy
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

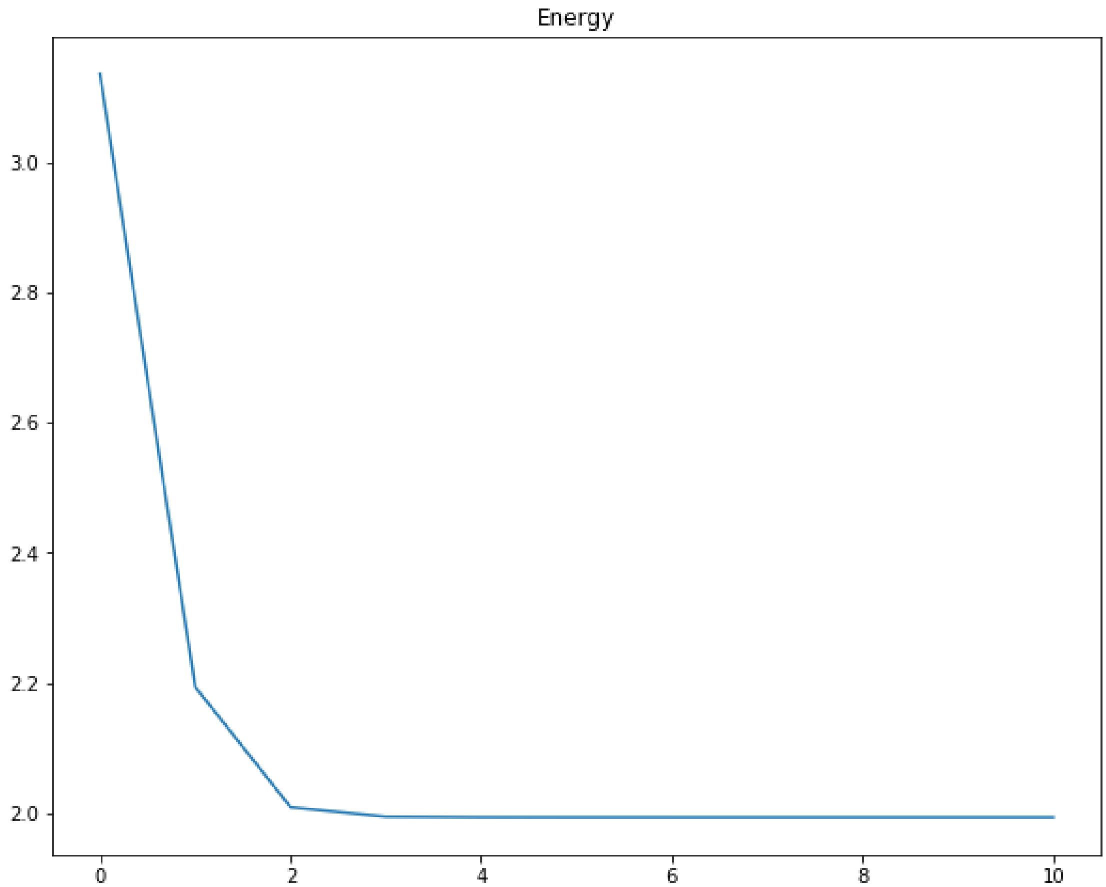
```
In [18]: initialize_label(2, 1)
```

```
3.1362979662055
2.193680680439046
2.008824139681163
1.9942392684086043
1.9934451880496744
1.99337619584828
1.993366456784945
1.9933651074811398
1.9933647852986864
1.9933646245657362
1.993364598306356
end



K =  2
Lambda =  1
```



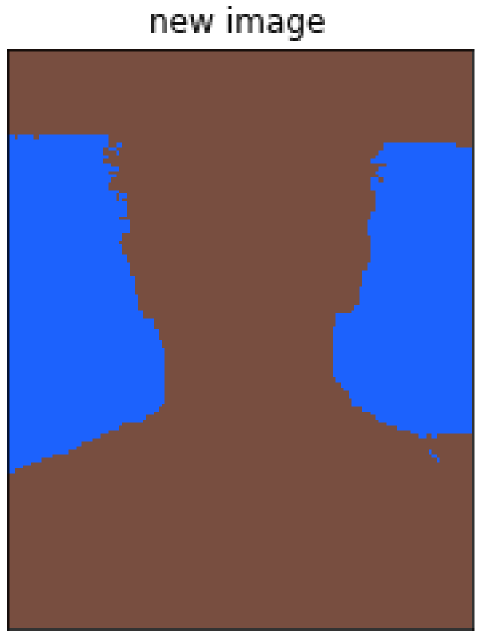


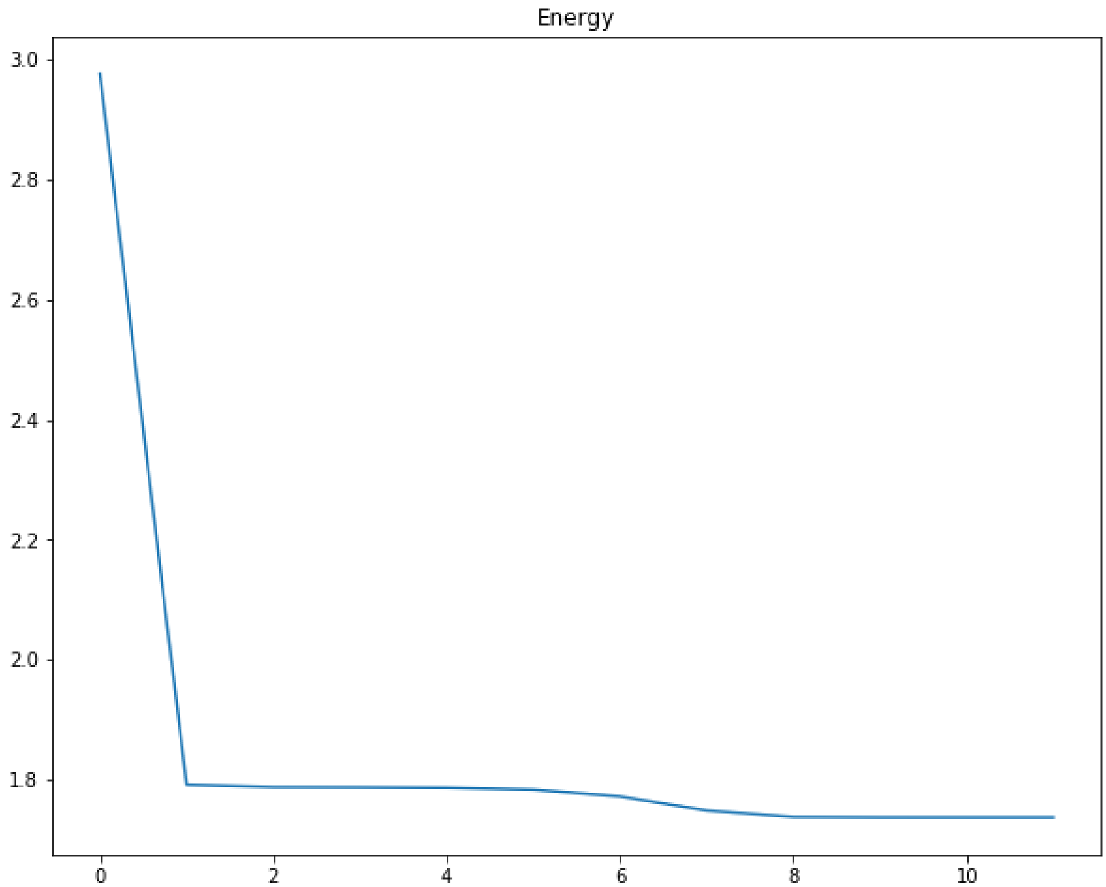

```
In [19]: initialize_label(2, 0.05)
```

```
2.976411675897365
1.791612769358359
1.7876993648264752
1.7874004750891592
1.7865789268928134
1.783225322752128
1.7720082357559261
1.7484291781399195
1.7377773732104185
1.7372393583103196
1.7371986034211093
1.7371972003493037
end
```

```
K =  2
```

Lambda = 0.05



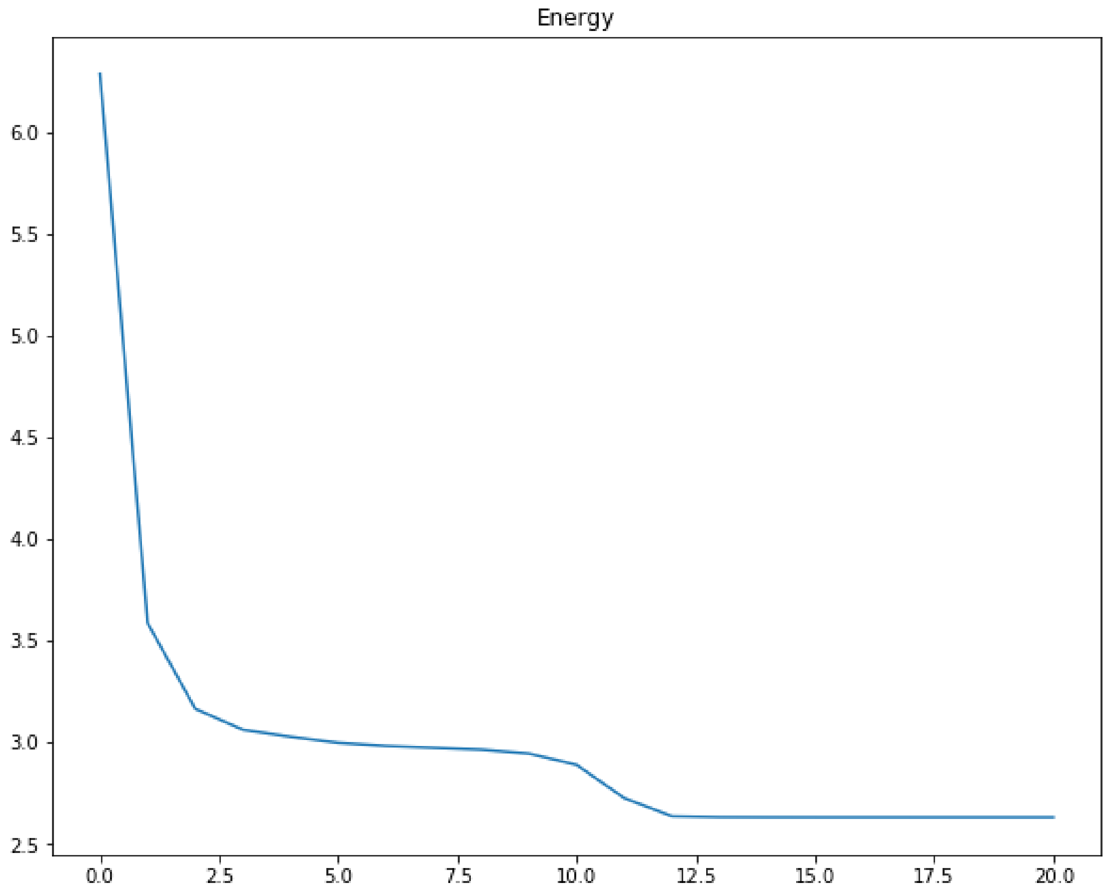



```
In [20]: initialize_label(4, 20)
```

```
6.287448485869407
3.5868092988362634
3.164817751642517
3.061575275208983
3.027719257071255
2.9979103038736272
2.982216832964148
2.97344668428856
2.9642179954459396
2.944827910126669
2.889237843806776
2.7256698337232343
2.6363838568295996
2.6321272993061795
2.631618185507257
2.631496051586129
2.6314412994798535
```

```
2.631412169694608
2.6314029994823005
2.631402177659895
2.6314020941736604
end
```

```
K =  4
Lambda =  20
```

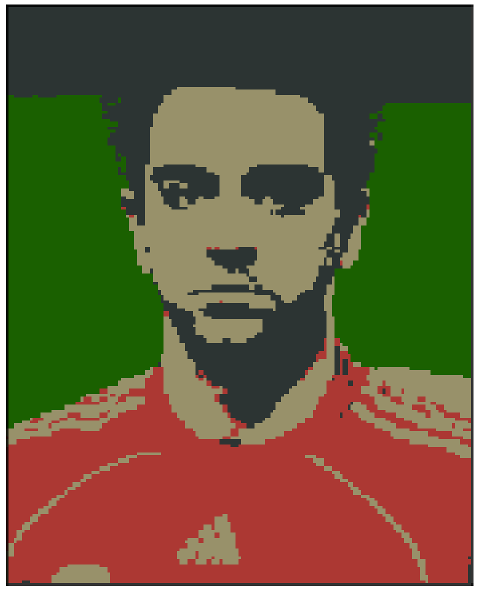


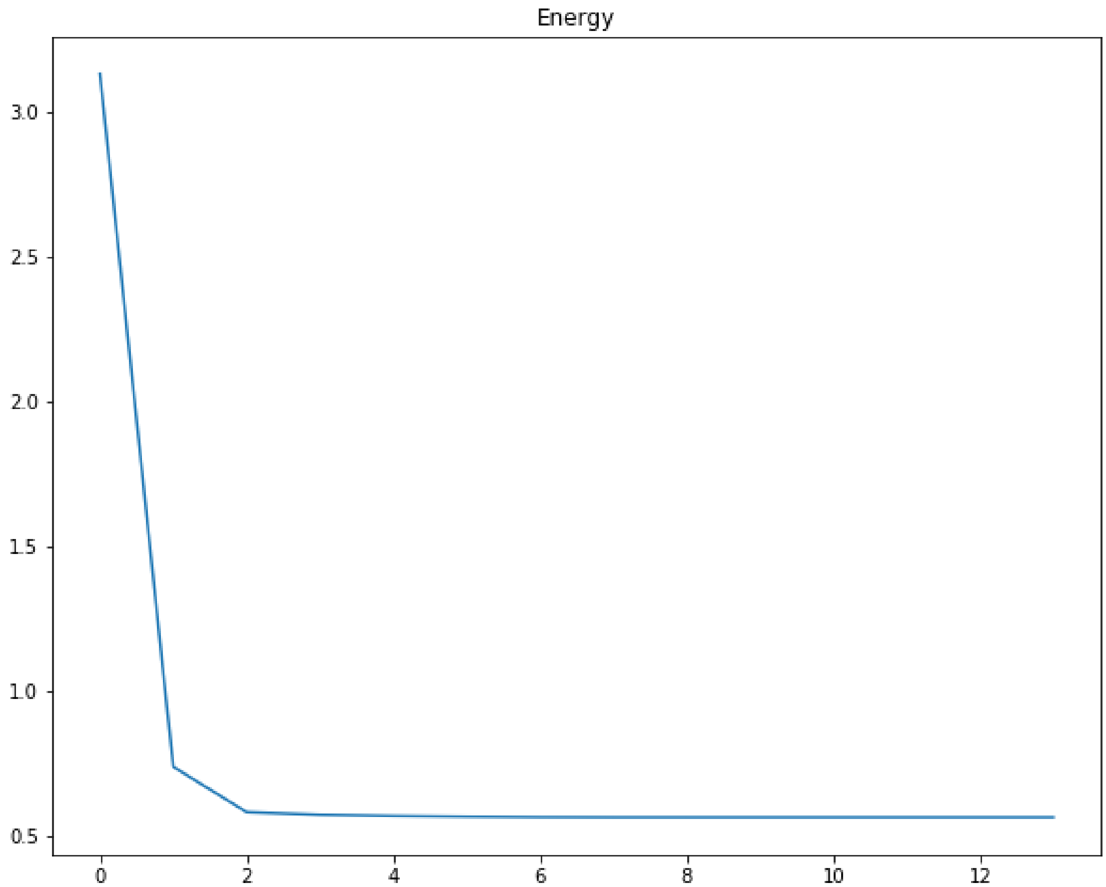
In [21]: initialize_label(4, 1)

```
3.1294671698367433
0.7388371536896199
0.582204139535475
0.5730115078563393
0.5691005426818783
0.5666217438765485
0.5649368676103249
0.5645007302990549
0.5644521482248168
0.5644444820772055
0.5644420269701531
0.5644418799559233
0.56444181424662
0.5644417661885497
end
```

```
K =  4
Lambda =  1
```

new image



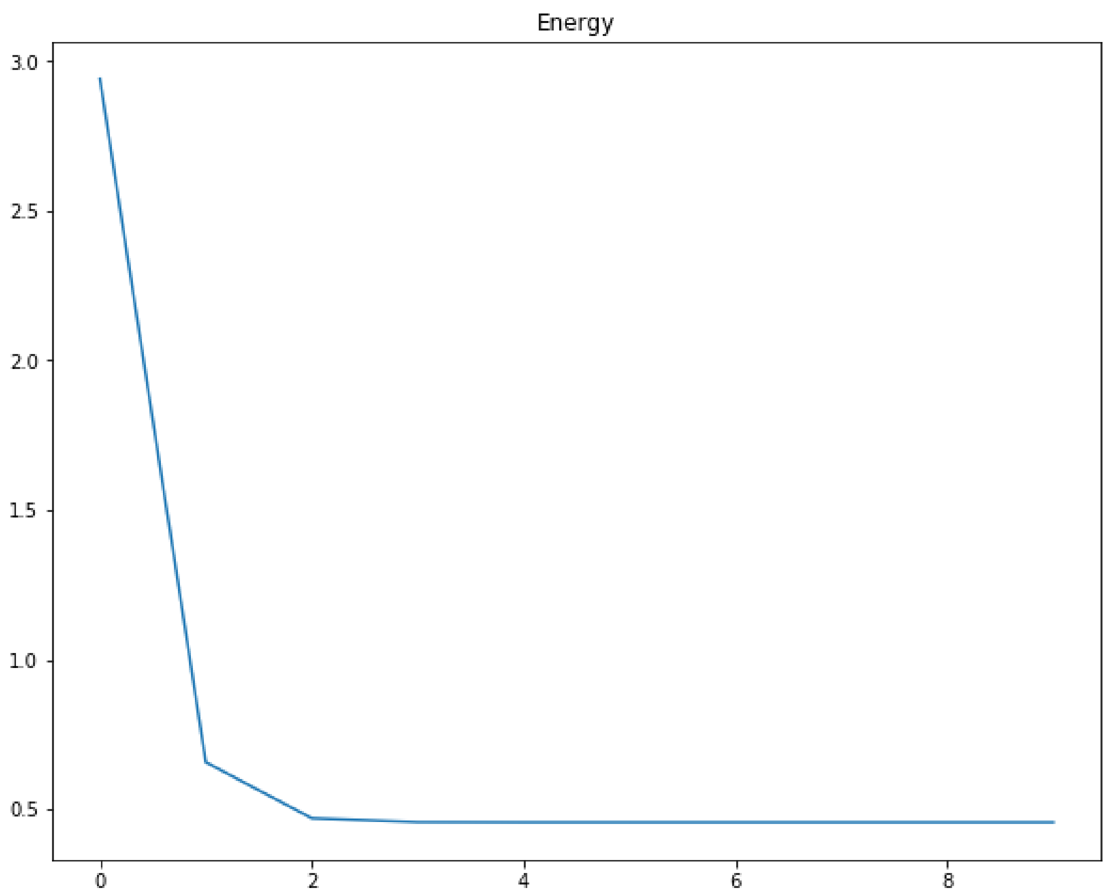



```
In [22]: initialize_label(4, 0.05)
```

```
2.9414551816245207
0.6571788002045369
0.4699625453096786
0.45698204525878816
0.4564760463360059
0.45645068208589945
0.45644861751245003
0.4564482939235302
0.45644824359107283
0.45644820547709514
end
```

```
K =  4
Lambda =  0.05
```


new image


Energy
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8

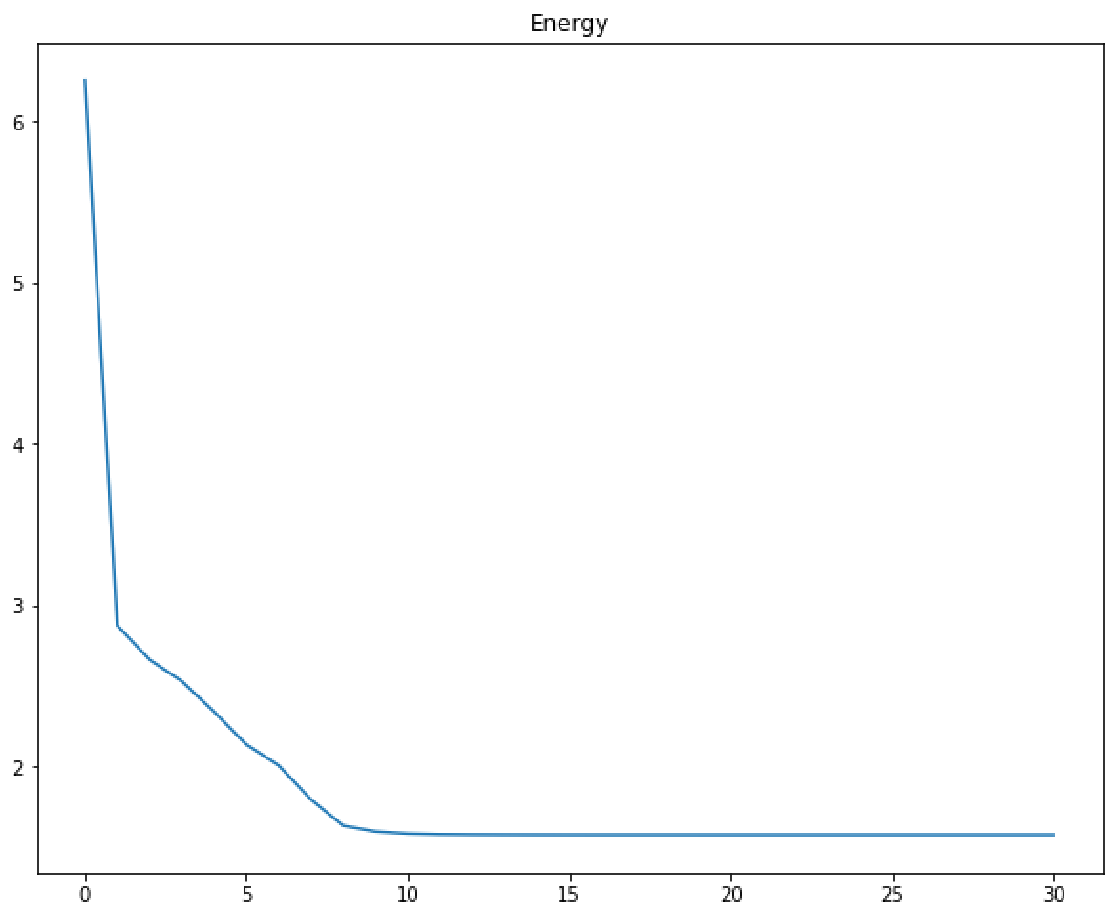
In [23]: initialize_label(6, 20)

```
6.257288590681303
2.875539725648612
2.664016183441916
2.5300732711583858
2.340487762266746
2.1373146184839658
2.008388013488495
1.796522300355282
1.6339923140800368
1.597626104129501
1.5846358244916356
1.580104696979479
1.578461513959601
1.57779383660354
1.5774411480419748
1.577263639989497
1.5771945374948473
1.577178505011662
1.5771670247078193
1.577159426661976
1.5771522335877566
1.577145153258903
1.5771399186605195
1.5771349070186766
1.5771309926628685
1.577126989943171
1.5771255205315389
1.5771245408545673
1.5771233540134875
1.5771226962528422
1.5771225594463414
end



K =  6
Lambda =  20
```


new image

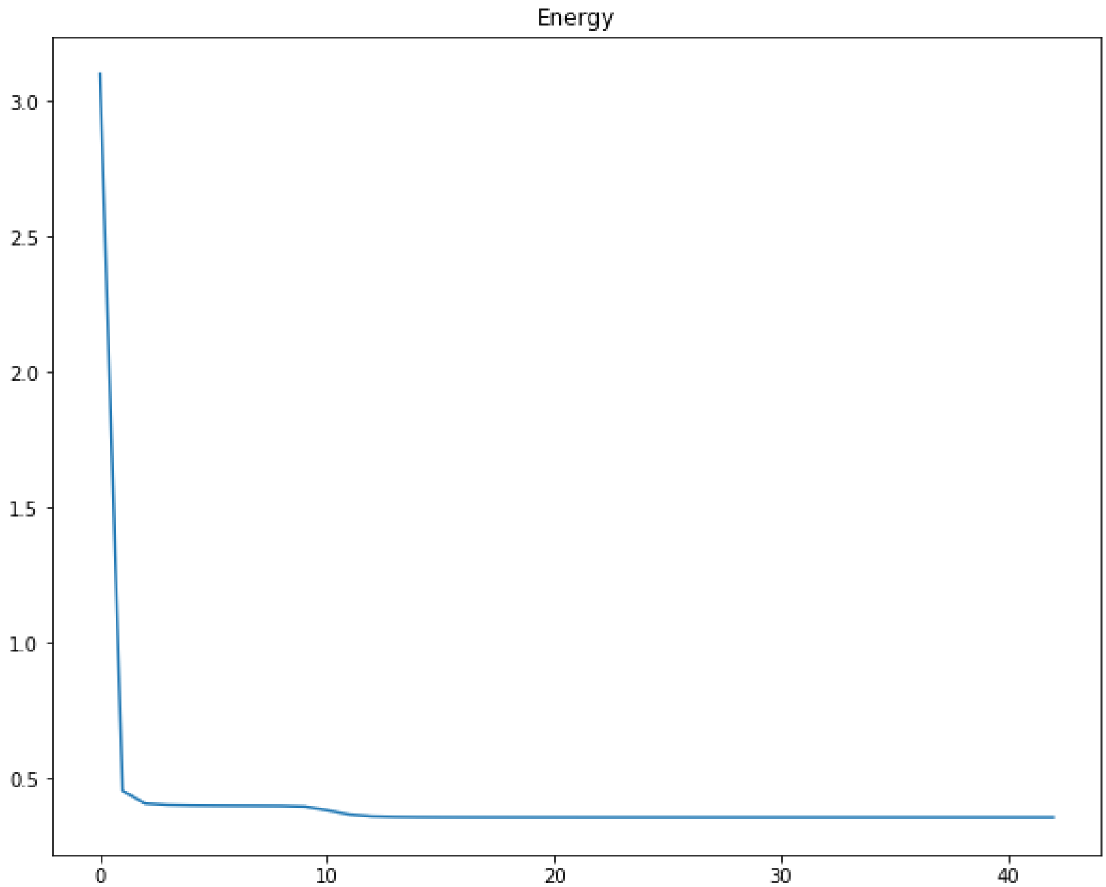

Energy
6
5
4
3
2
0
5
10
15
20
25
30

```
In [24]: initialize_label(6, 1)
```

```
3.101029932700333
0.45453102188342487
0.40761721484206376
0.40247183865655234
0.4007700153383817
0.40003065460196463
0.39954171039561764
0.3992259639709176
0.3987673938402135
0.39631490112263346
0.38311304487696024
0.36685759620702885
0.3601020448053854
0.3582148733534024
0.3575436212363974
0.35729230235557413
0.357172033341932
0.3571014902875592
0.3570710738972731
0.3570408774971237
0.3570088465279606
0.35698843442441425
0.35696823537305084
0.3569557547958221
0.35694581797380437
0.35694219005147404
0.3569376921862309
0.3569338899862083
0.3569298064225787
0.3569272801768182
0.356925689415535
0.3569247032800048
0.35692343232773793
0.3569228584236548
0.3569225846234277
0.3569220187847058
0.35692094203861136
0.35692044168093956
0.35691975800789877
0.35691848345992994
0.35691727370965765
0.35691650630426175
0.3569164701487643
```

```
end
```

```
K =  6
Lambda =  1
```



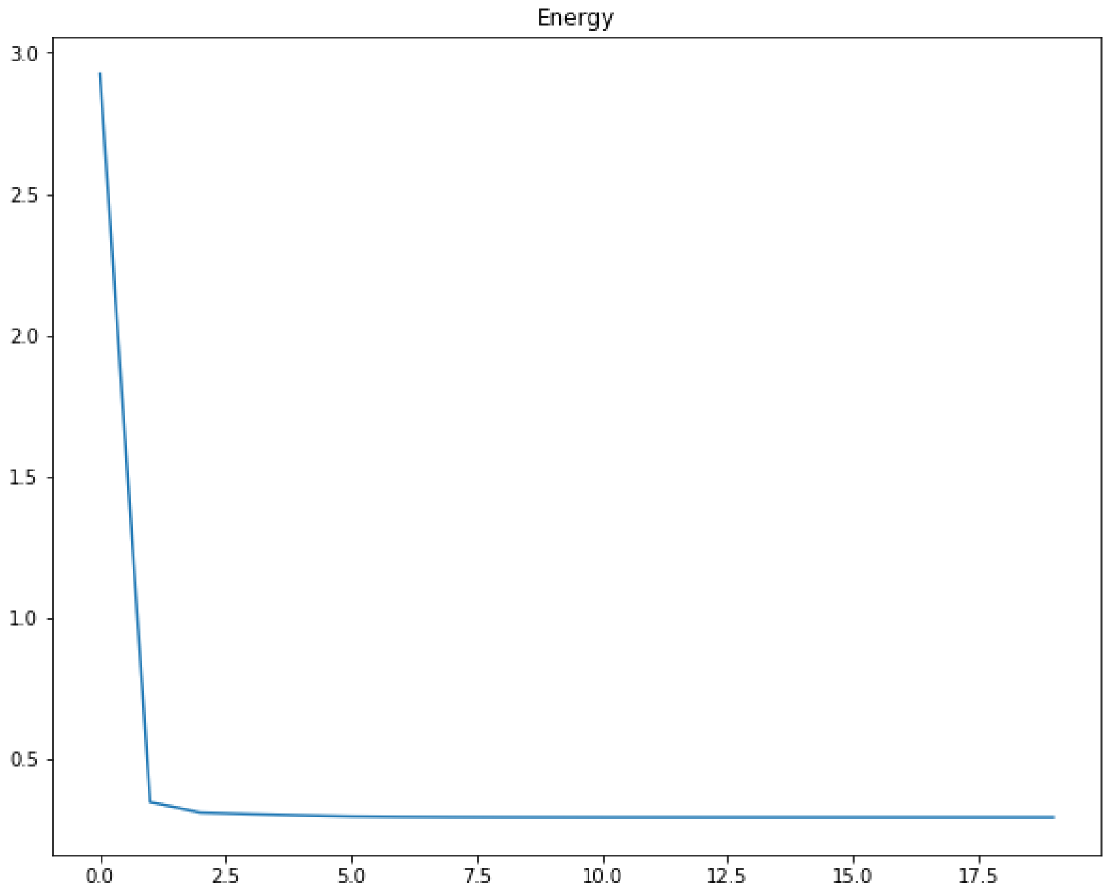

```
In [25]: initialize_label(6, 0.05)
```

```
2.9250572313914107
0.3484022805293294
0.31076427535322754
0.30536747403857156
0.30089366496775816
0.29714637493897467
0.29566782222055954
0.2949616106417537
0.2946834214220558
0.29454924105748326
0.29446137283062296
0.2944180799852889
0.29439822039606217
0.29439383159420274
0.294393014222521
0.29439279968410986
0.2943922268292611
```

```
0.2943919547994242
0.29439156359498797
0.2943915371291321
end
```

```
K =  6
Lambda =  0.05
```

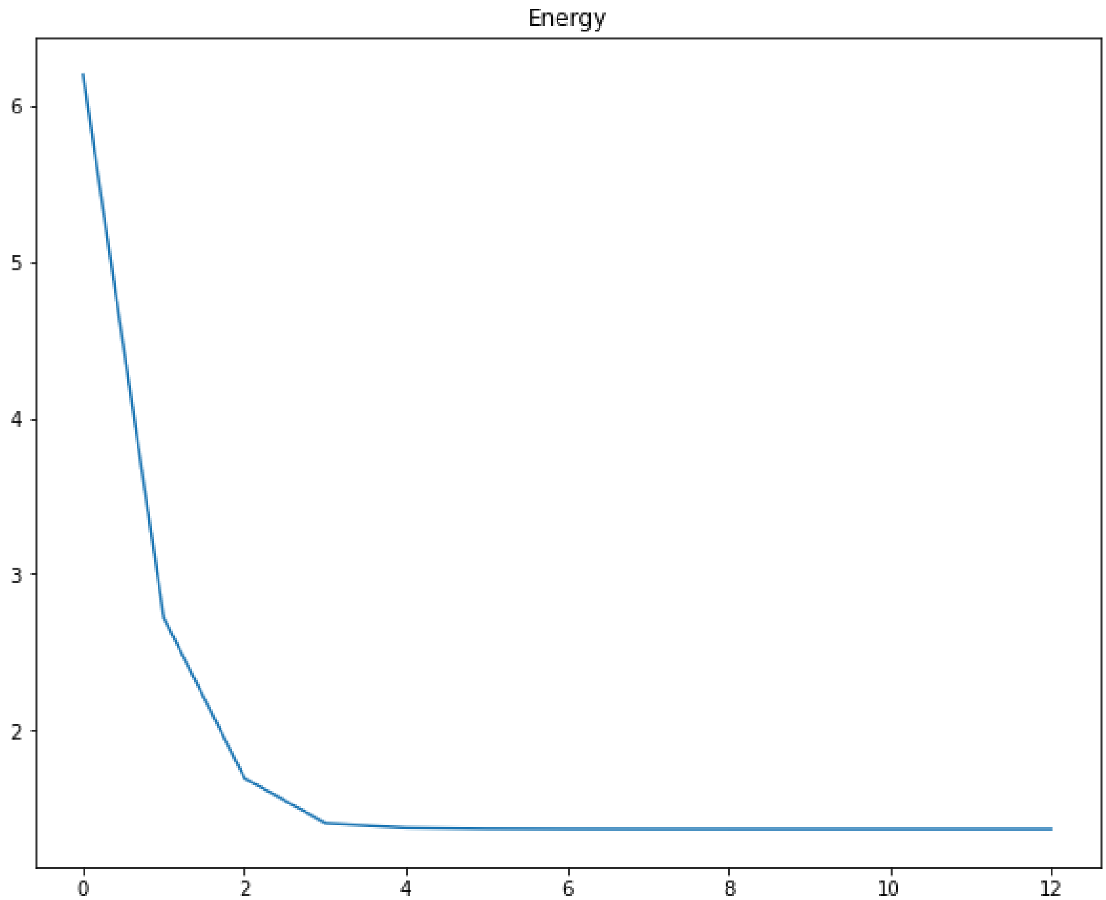
new image




```
In [26]: initialize_label(7, 20)
```

```
6.19615269073586
2.7201861228100688
1.6954192089069215
1.4075526426564993
1.3769957067317982
1.3707428902372376
1.3692105390036058
1.3688592274931757
1.3687510960732912
1.3687390274157891
1.3687358575443762
1.3687352986811065
1.368735239869986
end
```

```
K =  7
Lambda =  20
```

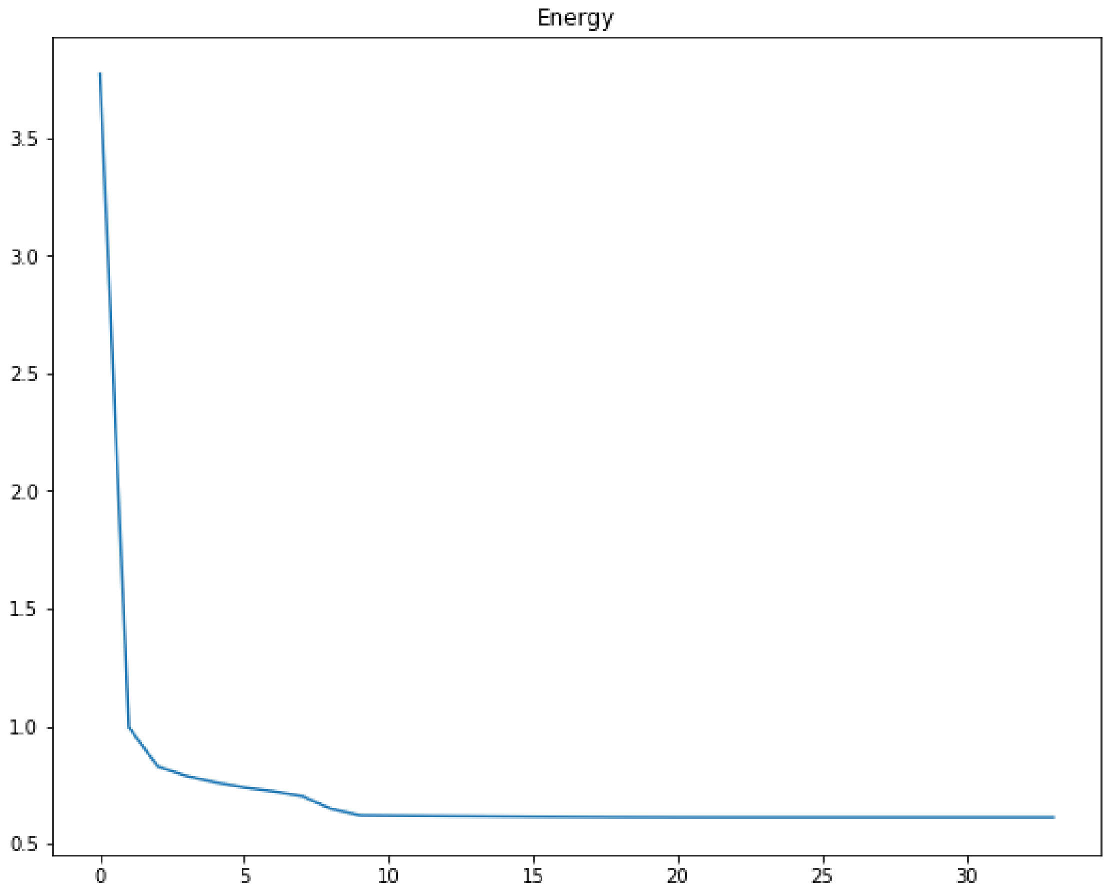



```
In [27]: initialize_label(7, 5)
```

```
3.772089986894705
0.9957851390280129
0.8299048483924044
0.7886509242219683
0.7618956884567452
0.740358892547814
0.7235445256243971
0.7030088491280728
0.649354948058015
0.6216751822551964
0.6204116752916897
0.6194494048535671
0.6185345676539746
0.6177420503443356
0.6167905067267242
0.6158872286721766
0.6152671976880179
```

```
0.6146760517587009
0.614078567253629
0.6137493081461359
0.6135365469007313
0.6134065708655726
0.6133623403183341
0.6133384173748163
0.613324788165119
0.6133203935313631
0.6133165885607373
0.6133112278193528
0.6133106913005233
0.6133104601875281
0.6133104030444588
0.6133103577402964
0.6133102896307115
0.6133102615092151
end
```

```
K =  7
Lambda =  5
```

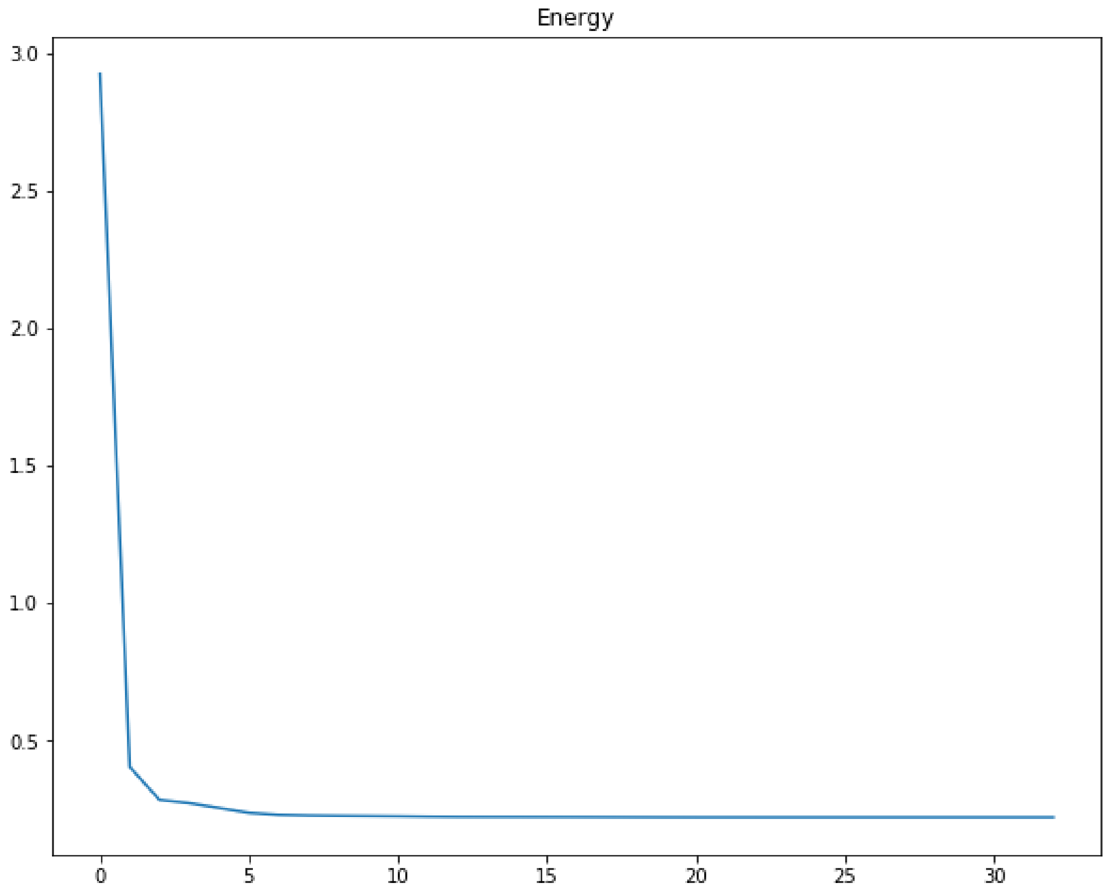



```
In [28]: initialize_label(7, 0.05)
```

```
2.9258360169197437
0.40400832058466807
0.28423522569829945
0.2724680640838834
0.25455062046704524
0.23704528704228597
0.22955538133558887
0.2272438101029319
0.22607136556827348
0.22534253283515812
0.2244502902997358
0.2231544362046385
0.22235246764410332
0.22213721268461034
0.2220317857142844
0.221882141728794
0.2217139305949875
```

```
0.221470735455555493
0.22119052488022248
0.2209900477586178
0.2208498928472801
0.22081661701364902
0.22080151335150427
0.22079104540466674
0.22078663556903036
0.22078425311781638
0.22078355745971176
0.220783329306708
0.22078308497734192
0.22078293277027028
0.2207826367640444
0.22078203383030792
0.2207817003698318
end
```

```
K =  7
Lambda =  0.05
```



Energy
3.0
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
30