EspcTM-demo

September 3, 2020

```
In [1]: #import all math libraries
        import numpy as np
        import numpy.fft as nf
        import matplotlib.pyplot as plt
        from matplotlib import cm
        from sklearn import linear_model
        from sklearn import decomposition
```

```
In [2]: #read dihedral angles data
        rama=np.loadtxt('rama_all.xvg')
        print(rama[0:10])
        print("shape of rama",np.shape(rama))
```

```
[[-135.888  -160.113 ]
 [-150.233   -99.0128]
 [ -70.0808  -11.9132]
 [ -57.4871  -14.104 ]
 [-129.121     7.0965]
 [ -82.5249  -24.4214]
 [-157.505   157.605 ]
 [-170.82    158.903 ]
 [-120.77    157.24  ]
 [-101.605  -174.711 ]]
shape of rama (1000000, 2)
```

```
In [3]: #calculate sine and cosine of backbone dihedral angles Phi and Psi
        feature=np.zeros([int(len(rama)/10),40])
        for i in range(len(feature)):
            for j in range(10):
                feature[i][j*4+0]=np.sin(rama[i*10+j][0]/180*np.pi)
                feature[i][j*4+1]=np.sin(rama[i*10+j][1]/180*np.pi)
                feature[i][j*4+2]=np.cos(rama[i*10+j][0]/180*np.pi)
                feature[i][j*4+3]=np.cos(rama[i*10+j][1]/180*np.pi)
        print("shape of feature",np.shape(feature))
```

```
shape of feature (100000, 40)
```

In [4]:
```
#normalization
feature_norm=np.zeros([int(len(rama)/10),40])
for i in range(40):
    ave=np.mean(feature[:,i])
    std=np.std(feature[:,i],ddof=1)
    feature_norm[:,i]=(feature[:,i]-ave)/std
```

In [5]:
```
#read potential energy data
energy=np.loadtxt('Scale_all.xvg')
print(energy[0:10])
print("shape of energy",np.shape(energy))
```

```
[[ 1.00000000e+00 -1.47027109e+05]
 [ 2.00000000e+00 -1.47431031e+05]
 [ 3.00000000e+00 -1.48044359e+05]
 [ 4.00000000e+00 -1.47358062e+05]
 [ 5.00000000e+00 -1.47179625e+05]
 [ 6.00000000e+00 -1.47591656e+05]
 [ 7.00000000e+00 -1.47060844e+05]
 [ 8.00000000e+00 -1.47151953e+05]
 [ 9.00000000e+00 -1.47441766e+05]
 [ 1.00000000e+01 -1.47800828e+05]]
shape of energy (100000, 2)
```

In [6]:
```
#define a function of  FFT and reverse FFT for noise reduction
def FFT(signal,cutoff):
    Y=[]
    for i in range(1,len(signal)):
        Y.append(signal[len(signal)-i])
    Y.append(signal[0])
    for i in range(len(signal)):
        Y.append(signal[i])

    comp_arr = nf.fft(Y)
    for i in range(len(comp_arr)):
        if i >cutoff and i < len(comp_arr)-cutoff:
            comp_arr[i] = 0
    Y2 = nf.ifft(comp_arr).real
    y2=Y2[int(len(Y)/2):]
    return y2
```

In [7]:
```
#calculate multiple correlation coefficient
def R(x,y):
    model=linear_model.LinearRegression()
    model.fit(x,y)
    q=sum( (y-model.predict(x))**2 )
    y_ave=np.mean(y)
    t=sum( (y-y_ave)**2 )
```

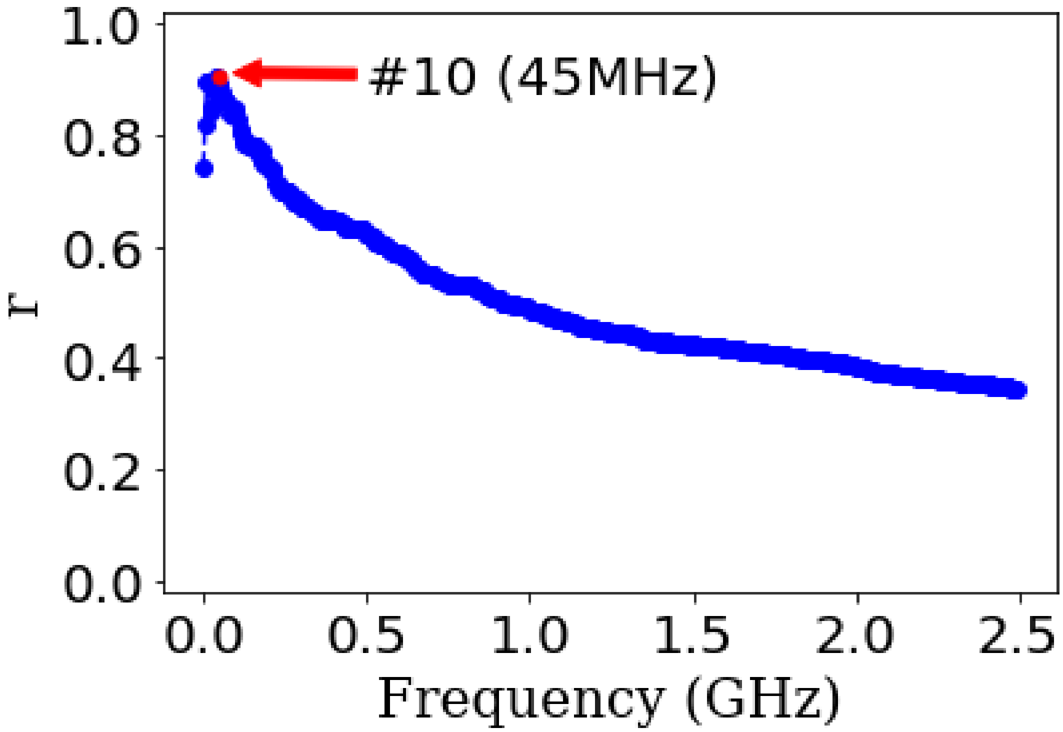
```
    r=np.sqrt(1-q/t)
    return r
```

In [8]:
```
#optimize frequency cutoff by multiple correlation coefficient
rr=[]
for cutoff in range(1,500):
    rr.append(    R( feature_norm,FFT(energy[:,1],cutoff) )     )
```

In [96]:
```
#find the optimize frequency cutoff which is 9,
#that means first 10 frequencies will be used.
print("Best cutoff",np.argmax(rr)+1)
fig, ax =plt.subplots()
font = {'family' : 'serif',
        'color'  : 'black',
        'weight' : 'normal',
        'size'   : 20,
        }
plt.plot(rr,"bo--")
xticks=range(0,500+100,100);ax.set_xticks(xticks)
xticklabels=['0.0','0.5','1.0','1.5','2.0','2.5']
ax.set_xticklabels(xticklabels, fontsize=20)

min_y=0;max_y=1
plt.ylim(min_y-max_y/50, max_y+max_y/50)
yticks=np.arange(0,max_y+max_y/5.0,max_y/5.0)
ax.set_yticks(yticks);yticklabels=['0.0','0.2','0.4','0.6','0.8','1.0']
ax.set_yticklabels(yticklabels, fontsize=20)
plt.xlabel('Frequency (GHz)', fontdict=font)
plt.ylabel("r", fontdict=font)

plt.plot(9,rr[9-1], 'o', markersize=5, linewidth=2.5,color='red')

ax.annotate('#10 (45MHz)', xy=(15, 0.91), xytext=(100, 0.95),
            arrowprops=dict(arrowstyle="simple",facecolor='red',edgecolor='red'),
            fontsize=20,horizontalalignment='left', verticalalignment='top')
```

```
Best cutoff 9
```

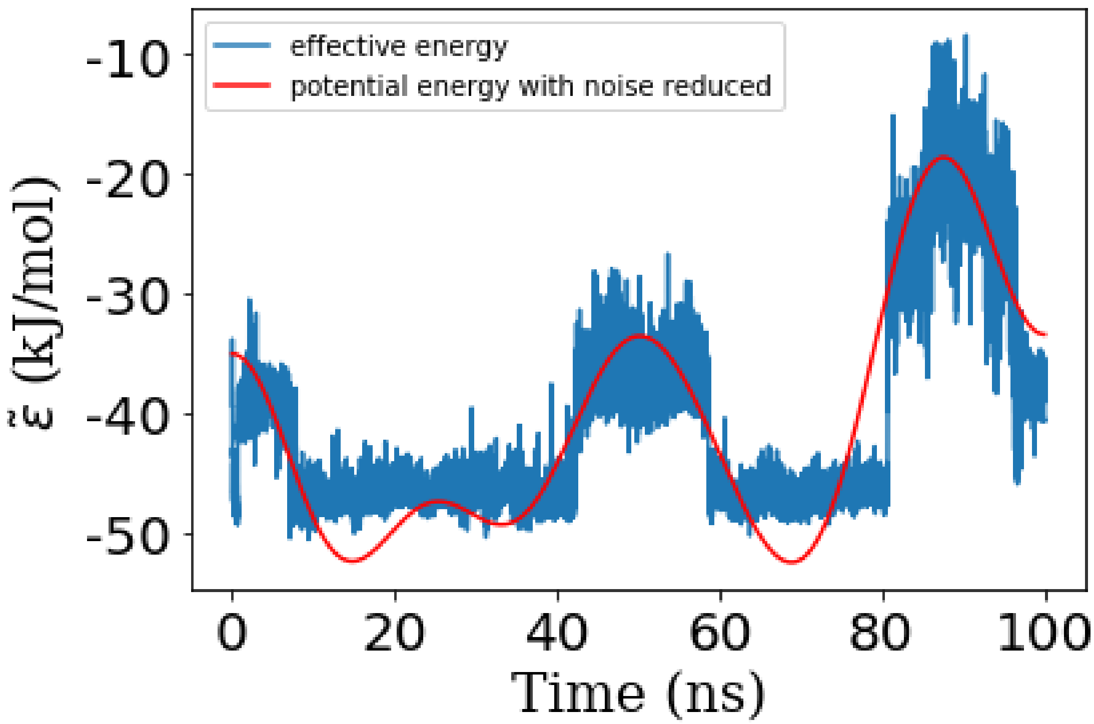
Out[96]:
```
Text(100, 0.95, '#10 (45MHz)')
```



In [94]:
```
#generate effective energy
model=linear_model.LinearRegression()
model.fit(feature_norm,FFT(energy[:,1],np.argmax(rr)+1))

fig, ax =plt.subplots()
font = {'family' : 'serif',
        'color'  : 'black',
        'weight' : 'normal',
        'size'   : 20,
        }
plt.plot(model.predict(feature_norm),label="effective energy")   #effective energy
#potential energy with noise reduced
plt.plot(FFT(energy[:,1],np.argmax(rr)+1),'r',label="potential energy with noise redu

xticks=range(0,100001,20000);ax.set_xticks(xticks);xticklabels=range(0,101,20);
ax.set_xticklabels(xticklabels, fontsize=20)

yticks=range(-50-147400,-10-147400+1,10)
ax.set_yticks(yticks)


yticklabels=range(-50,-10+1,10)
ax.set_yticklabels(yticklabels, fontsize=20)
```

```
ax.set_xlabel('Time (ns)', fontdict=font)
ax.set_ylabel(r'$\~ \epsilon$ (kJ/mol)', fontdict=font)
plt.legend(fontsize=10)
```

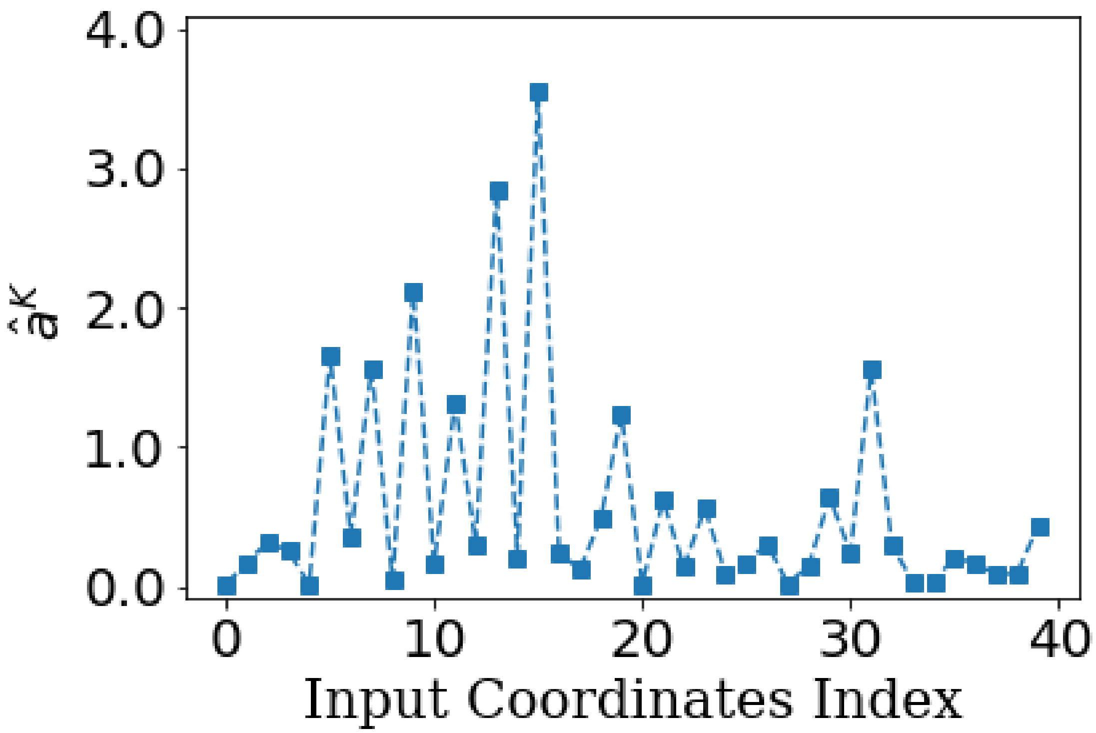
Out[94]: <matplotlib.legend.Legend at 0x2b714fadc4e0>



In [64]:
```
fig, ax =plt.subplots()
font = {'family' : 'serif',
        'color'  : 'black',
        'weight' : 'normal',
        'size'   : 20,
        }
plt.plot(np.abs(model.coef_),"s--")
xticks=range(0,40+10,10)
ax.set_xticks(xticks)
xticklabels=range(0,40+10,10)
ax.set_xticklabels(xticklabels, fontsize=20)

min_y=0.0; max_y=4.0
plt.ylim(min_y-max_y/50, max_y+max_y/50)
yticks=np.arange(0,max_y+max_y/4.0,max_y/4.0)
ax.set_yticks(yticks)
yticklabels=np.arange(0,max_y+max_y/4.0,max_y/4.0)
ax.set_yticklabels(yticklabels, fontsize=20)
```

```
plt.xlabel("Input Coordinates Index", fontdict=font)
plt.ylabel(r"$\hat a ^{K}$", fontdict=font)
```

Out[64]: Text(0, 0.5, '$\\hat a ^{K}$')



In [12]:
```
#rescale all features
feature_norm_rescaled=np.zeros([int(len(rama)/10),40])
for i in range(40):
    feature_norm_rescaled[:,i]=feature_norm[:,i]*np.abs(model.coef_[i])
```

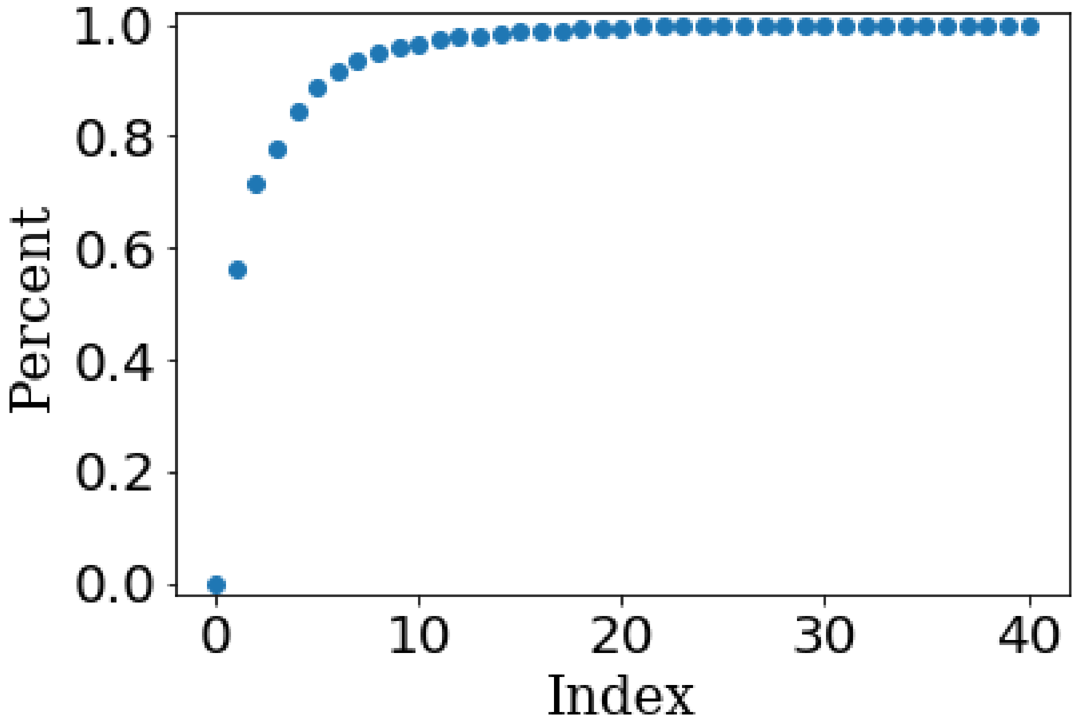
In [66]:
```
#PCA
pca_test=decomposition.PCA()
pca_test.fit( np.mat(feature_norm_rescaled) )

sum_of_eigenvalue=[]
for i in range(len(pca_test.explained_variance_ratio_)+1):
    sum_of_eigenvalue.append( sum(pca_test.explained_variance_ratio_[:i]) )

for i in range(len(sum_of_eigenvalue)):
    if sum_of_eigenvalue[i] >=0.9:
        n=i
        break
```

```
print("dimension:",n)
pca = decomposition.PCA(n_components=n,copy=True,whiten=False)
pca.fit( np.mat(feature_norm_rescaled) )
feature_mapped=pca.transform( np.mat(feature_norm) )
print("shape of feature_mapped",np.shape(feature_mapped))
```

```
dimension: 6
shape of feature_mapped (100000, 6)
```

In [77]:
```
fig, ax =plt.subplots()
font = {'family' : 'serif',
        'color'  : 'black',
        'weight' : 'normal',
        'size'   : 20,
        }
plt.plot(sum_of_eigenvalue,"o")
xticks=range(0,40+10,10)
ax.set_xticks(xticks)
xticklabels=range(0,40+10,10)
ax.set_xticklabels(xticklabels, fontsize=20)

min_y=0.0; max_y=1.0
plt.ylim(min_y-max_y/50, max_y+max_y/50)
yticks=np.arange(0,max_y+max_y/5.0,max_y/5.0)
ax.set_yticks(yticks)
yticklabels=np.round(np.arange(0,max_y+max_y/5.0,max_y/5.0),1)
ax.set_yticklabels(yticklabels, fontsize=20)

plt.xlabel("Index", fontdict=font)
plt.ylabel("Percent", fontdict=font)
```

Out[77]:
```
Text(0, 0.5, 'Percent')
```



In [78]:
```
# define a function to average the trejectory for reduction of computing resources
def window_ave(x,length):
    m,n=np.shape(x)
    y=np.zeros( [int(m/length),n] )
    for j in range(n):
        for i in range(int(m/length)):
            y[i,j]=np.mean( np.split(x[:,j],int(m/length))[i] )
    return y
```

In [79]:
```
#define a function to calculate the similarity_matrix
def similarity_matrix(data):
    m,n=np.shape(data)
    sum_t1=0
    sum_t2=0
    sum_t1t2=0
    M=np.zeros([m,m])
    for i in range(m):
        for j in range(m):
            for k in range(n):
                sum_t1+=data[i,k]*data[i,k]+1
                sum_t2+=data[j,k]*data[j,k]+1
                sum_t1t2+=data[i,k]*data[j,k]+1
```

```
                cov_t1t2=sum_t1t2/np.sqrt(sum_t1*sum_t2);
                sum_t1=0;
                sum_t2=0;
                sum_t1t2=0;
                M[i,j]=cov_t1t2;
                cov_t1t2=0;
        return M
```

```
In [83]: #average features
         feature_mapped_averaged=window_ave(feature_mapped,200)
         print("shape of feature_mapped_averaged",np.shape(feature_mapped_averaged))
```

```
shape of feature_mapped_averaged (500, 6)
```

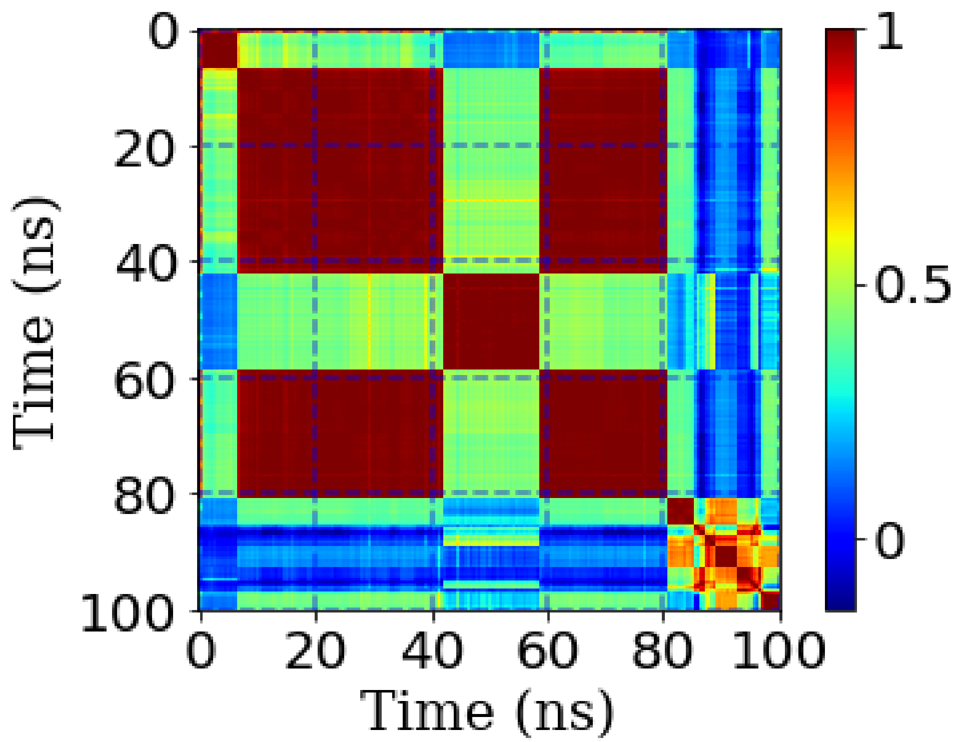
```
In [84]: #calculate the similarity_matrix
         MM=similarity_matrix(feature_mapped_averaged)
         print("shape of MM",np.shape(MM))
```

```
shape of MM (500, 500)
```

```
In [92]: #visualize similarity matrix
         print(np.shape(MM))
         fig, ax = plt.subplots()
         cax = ax.imshow(MM, interpolation='nearest', cmap=cm.jet)
         cbar = fig.colorbar(cax, ticks=[0,0.5, 1])
         cbar.ax.set_yticklabels(['0','0.5', '1'],fontsize=20)
         plt.grid(color='b', alpha=0.5, linestyle='dashed', linewidth=1.5)

         N=len(MM);Start_time=0;End_time=100
         xticks=range(0,500+1,100)
         ax.set_xticks(xticks)
         xticklabels=range(0,100+1,20)
         ax.set_xticklabels(xticklabels, fontsize=20)
         yticks=range(0,500+1,100)
         ax.set_yticks(yticks)
         yticklabels=range(0,100+1,20)
         ax.set_yticklabels(yticklabels, fontsize=20)
         plt.xlabel('Time (ns)', fontdict=font)
         plt.ylabel('Time (ns)', fontdict=font)
```

```
(500, 500)
```

```
Out[92]: Text(0, 0.5, 'Time (ns)')
```



0
20
40
60
80
100
Time (ns)
1
0.5
0