Final_project

February 24, 2021

1 Final task: Dimensionality reduction via principal component analysis

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import animation, rc
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

from IPython.display import HTML
import random
import copy
import numpy as np
import scipy as scy

from tqdm.notebook import trange, tqdm
import time
from scipy.stats import maxwell

#used for the video
import subprocess
import glob
import os
```

1.1 Introduction:

The analysis of highly dimentional systems described by a set of correlated coordinate can be very tedous, we therefore aim to simplify the system in such a way that that the anaylsis becomes computationally affordable and that the simplified model still describes the system in a revelent way. In this simulation we are here looking at dimentionality reduction using Principal componant analysis (PCA). The initial correlated set of corrdinates are modified by calculating the covarience matrix, diagonalising it, and obtaining it's eingenvectors and eingenvalues. With this transformation we order the coordinated by oder of magnitude of their eingen values. The eigenvectors give the new coordinates and the ones with higher eingenvalues are the ones along which the varience of the system is highter, and therefore the ones that describe the pricipal motion of the system. With a system with n sets of coordinates we can now choose the m (with $m << n$) most important ones

and analyse the system in lower dimentions with this new set of coordinates.

This simulation is runed with a relatively simple system (2D) and aims to show how PCA technic can help reduce dimentions by at the same time preserving the main characteristics of the system. We are here studying a 2D potential and obtaining out data of the energy surface by implementing a Monte Carlo method with Metropolis algorithm. We will study the difference of the data in the inital coordinates (x and y) and in the new set calculated from the covarience martix σ.

1.2 Task I: Implementation

Monte Carlo (MC) method with Metropolis algorithm that samples a 2D energy surface with:

$$U(x,y) = k_b T(0.28(0.25(a \cdot x + b \cdot y)^4 + 0.1(a \cdot x + b \cdot y)^3 - 3.24(a \cdot x + b \cdot y)^2 + 6.856(a \cdot y - b \cdot x)^2) + 3.5)$$

with $a = 0.809$ and $b = 0.588$.

- Calculates the potential energy given values of x and y using the above equation.

-> Takes : x, y

-> Returns : U(x,y)

[2]:
```
def Potential_energy (x,y):
    a = 0.809
    b = 0.588

    ax_plus_by = a*x + Dimensionalityb*y
    ay_minus_bx =a*y - b*x

    U = k_b * Temp * (0.28*( 0.25*ax_plus_by**4 + 0.1*ax_plus_by**3 - 3.
↪24*ax_plus_by**2 + 6.856*ay_minus_bx**2)+3.5)
    return U
```

- Metropolis algorithm :

  From our previous position (x,y) of sampling with choose a other position at a set distance dR but with random orientation.

  - If the potential is lower at this new location we sample there.
  - If it is higher we either discard the move or accept it with a certain probability depending of the difference of energy between the point positions and a ranom number.

This results in a sampling trajectory that will both tend to go towards the mininmum points of energy but htat can also go over barriers of potential, with a probability proportional to the gradient of the barrier. This allow us to find sevral minimas, as long as the sampling time is large engough.

-> Takes : (x,y) last sampling position

-> Returns : True/False (if move if accepted of not), new/same position of sampling, potential at this position

[3]:
```
def Move(x,y):
    #Calculate U1
    U_1 = Potential_energy(x,y)

    #Move by dR = 0.01 nm in a random direction
    angle = np.random.uniform(0,2*np.pi)
    dx = np.sin(angle)*dR
    dy = np.cos(angle)*dR

    #Calculate U2
    U_2 = Potential_energy(x+dx, y+dy)

    if U_1 > U_2:
        #accept move
        return True, x+dx , y+dy, U_2

    else :
        P = np.exp(-(U_2 - U_1)/(k_b*Temp))
        q = np.random.uniform(0,1)
        if q < P:
            #accept move
            return True, x+dx , y+dy, U_2
        else :
            #Discard move
            return False, x, y, U_1
```

1.3 Task II: Simulation

[4]:
```
Temp = 300 #K

#Step size
dR = 0.01 #nm

#Initial position of the first sampling
x_0 = 2
y_0 = 2

# Number of accepted mooves
Nbr_MC = 2000000

k_b = 1.380649e-23 #J.K-1
Na = 6.02214086e23 #mol-1
```

- Monte carlo simulation (with Metropolis algorithm included in the funcion 'Move')

-> Returns : x_list, and y_list of the position of the sampling over time and U_list which correspong to the value of the potential at those positions.

[ ]:
```
#Lists used to strore the trajectory and the potential along it.
x_list = [x_0] + [0]*Nbr_MC
y_list = [y_0] + [0]*Nbr_MC
U_list = [Potential_energy(x_0,y_0)] + [0]*Nbr_MC

#Number of accepted moves
move_Nbr = 0

while move_Nbr < Nbr_MC :

    #Prints the percentage of mooves reasiled out of Nbr_MC
    if (100*move_Nbr/Nbr_MC)%5 == 0 :
        print(str(100 * move_Nbr/Nbr_MC)[:3])

    # Metropolis algorithm
    check_value, x, y, U = Move( x_list[move_Nbr], y_list[move_Nbr])

    #If check value = True the move has been accepted and we perform the sample.
    # we store the data of thetrajectory in the lists. Otherwise we pass, staying
    # at the same position.
    if check_value == True :
        move_Nbr += 1
        x_list[move_Nbr] = x
        y_list[move_Nbr] = y
        U_list[move_Nbr] = U

print('Done')
```

```
0.0
5.0
10.
15.
20.
```

- Saves the data from the simulation or retreive data from the simulation run earlier

[ ]:
```
#np.savez('save2.npz', x_list, y_list, U_list)
```

[5]:
```
npzfile = np.load('save.npz')
x_list = npzfile['arr_0']
y_list = npzfile['arr_1']
U_list = npzfile['arr_2']
```

Calculating the minimum

Seeing the plot of the 2D energy surface with can distinguish 2 minimas. In the following cell we calculate their exact position and depth.

[6]:
```
U_list_min1 = [10]*len(x_list)
U_list_min2 = [10]*len(x_list)
for ind in trange(len(x_list)):
    #First minima at x < 0 :
    if x_list[ind] < 0 :
        U_list_min1[ind] = U_list[ind]
    #Second minima at x > 0 :
    else :
        U_list_min2[ind] = U_list[ind]

U_min_1 = min(U_list_min1)
min_ind_1 = U_list_min1.index(U_min_1)
U_min_2 = min(U_list_min2)
min_ind_2 = U_list_min2.index(U_min_2)
```

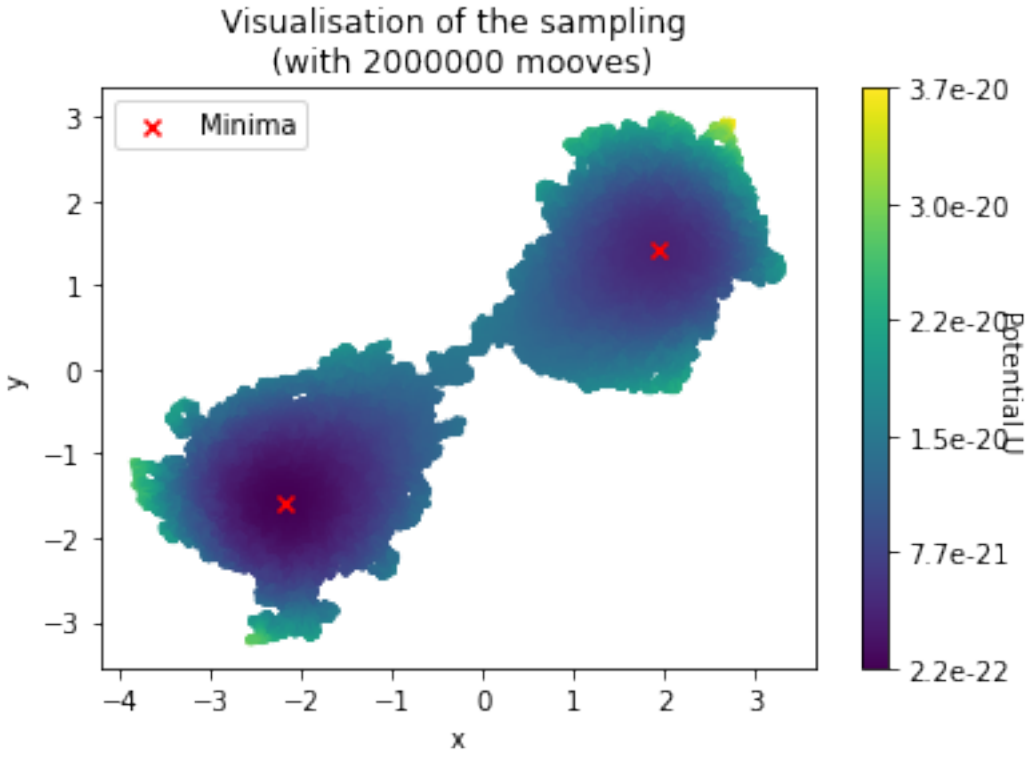
```
HBox(children=(FloatProgress(value=0.0, max=2000001.0), HTML(value='')))
```

Plot of the data

[7]:
```
plt.scatter(x_list, y_list, c = U_list, s=3)
plt.scatter([x_list[min_ind_1],x_list[min_ind_2]],␣
 ↪[y_list[min_ind_1],y_list[min_ind_2]], c = 'red', marker='x',label='Minima')

plt.xlabel('x')
plt.ylabel('y')
t = 'Visualisation of the sampling \n (with ' + str(Nbr_MC) + ' mooves)'
plt.title(t)
plt.legend()

cbar = plt.colorbar()

U_ticks = np.linspace(min(U_list), max(U_list), 6)
U_ticks = [str(U_ticks[i])[0:3] + str(U_ticks[i])[-4:] for i in␣
 ↪range(len(U_ticks))]
cbar.ax.set_yticklabels(U_ticks)

cbar.set_label('Potential U', rotation=270)

print('Minimum 1 at (x,y) = (', x_list[min_ind_1],y_list[min_ind_1], ') with U␣
 ↪= ', U_min_1, 'J')
print()
print('Minimum 2 at (x,y) = (', x_list[min_ind_2],y_list[min_ind_2], ') with U␣
 ↪= ', U_min_2, 'J')
```

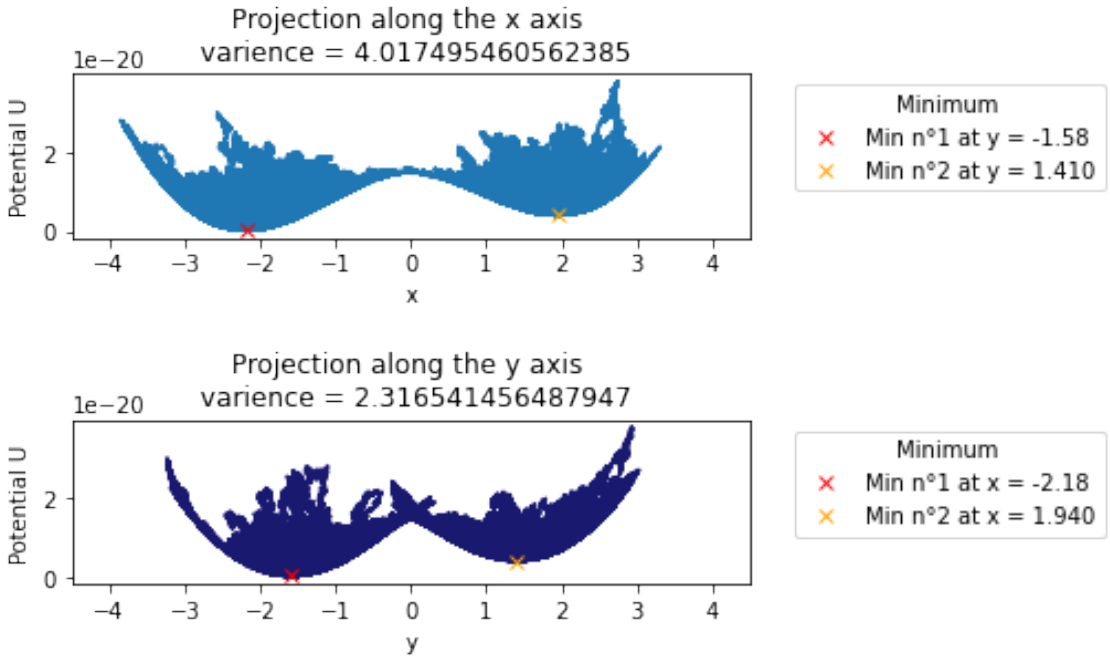
```
Minimum 1 at: ( -2.183668404027096 -1.5881000022882406 ) with U =
2.2978051441872487e-22
```

```
Minimum 2 at: ( 1.9408301256758511 1.4107920871090323 ) with U =
4.075809248912603e-21
```



[9]:
```
# Projection of the data along the x axisDimensionality
plt.subplot(211)

#Trajectory :
plt.plot(x_list, U_list)

#Indicates the mininmas:
MIN1_X = 'Min n°1 at y = ' + str(y_list[min_ind_1])[0:5]
MIN2_X = 'Min n°2 at y = ' + str(y_list[min_ind_2])[0:5]
plt.plot([x_list[min_ind_1]], [U_list[min_ind_1]],c = 'red', marker='x',␣
 ↪linestyle='None',label= MIN1_X )
plt.plot([x_list[min_ind_2]], [U_list[min_ind_2]],c = 'orange', marker='x',␣
 ↪linestyle='None',label= MIN2_X )

#Titles and labels for the plot :
t = 'Projection along the x axis \n varience = ' + str(np.var(x_list))
plt.title(t)
plt.xlim(-4.5, 4.5)
```

```
plt.xlabel('x')
plt.ylabel('Potential U')
plt.legend(title='Minimum ', bbox_to_anchor=(1.05, 1), loc='upper left')




# Projection of the data along the y axis
plt.subplot(212)

#Trajectory :
plt.plot(y_list, U_list, c = 'midnightblue')

#Indicates the mininmas:
MIN1_Y = 'Min n°1 at x = ' + str(x_list[min_ind_1])[0:5]
MIN2_Y = 'Min n°2 at x = ' + str(x_list[min_ind_2])[0:5]
plt.plot([y_list[min_ind_1]], [U_list[min_ind_1]],c = 'red', marker='x',␣
 ↪linestyle='None',label= MIN1_Y )
plt.plot([y_list[min_ind_2]], [U_list[min_ind_2]],c = 'orange', marker='x',␣
 ↪linestyle='None',label= MIN2_Y )

#Titles and labels for the plot :
t = 'Projection along the y axis \n varience = ' + str(np.var(y_list))
plt.title(t)
plt.xlim(-4.5, 4.5)
plt.xlabel('y')
plt.ylabel('Potential U')
plt.legend(title='Minimum ', bbox_to_anchor=(1.05, 1), loc='upper left')


#General adjustments :
plt.tight_layout()
plt.subplots_adjust(bottom=0.1, right=0.8, top=0.9)
plt.show()
```



We can clearly see the two minimas in the two projections, both corrdinates hold information about the main characteristic of the energy surface. Alough the varience along x is 1.7 times larger then the one along y, it is only describing 63.4 % of the total varience.

1.4 Task III: Sampling

Free energy :

$$\Delta G(x,y) = -k_b T ln(P(x,y))$$
$$= (U2 - U1) = \Delta U$$

With :

$$ln(P) = -\frac{(U2 - U1)}{k_b T}$$

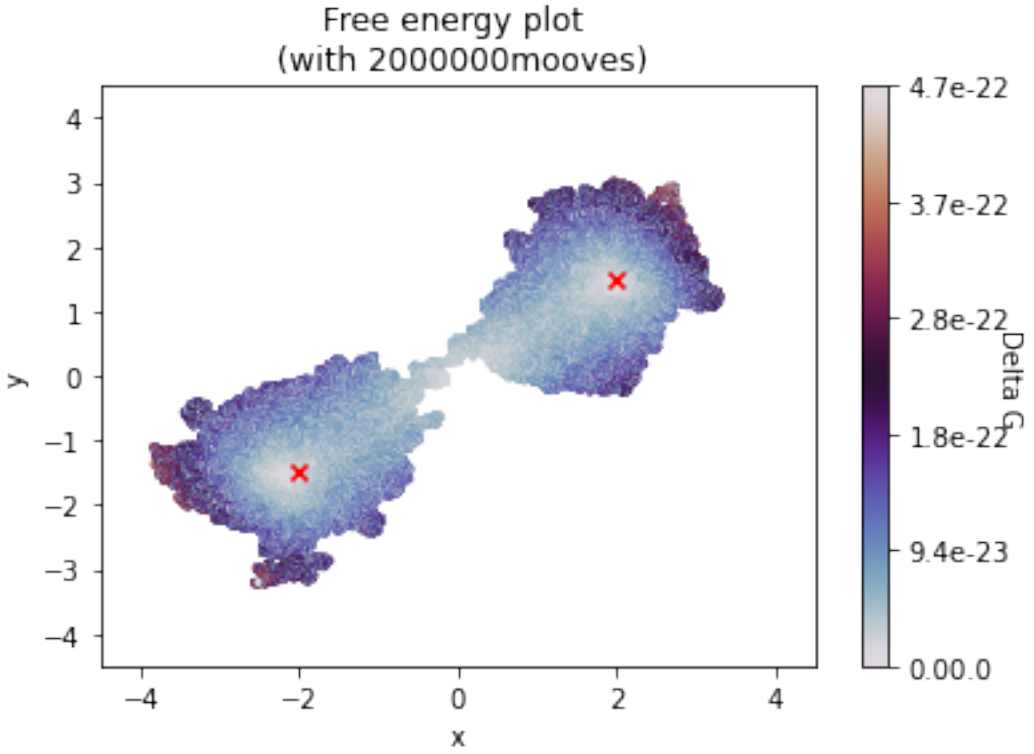
The plot will then show us the gradien of the energy surface. In the cell below we calculate the free energy ΔG for our trajectory.

[10]:
```
lnP_list = [0]* Nbr_MC
for i in trange (1, Nbr_MC):
    U_1 = Potential_energy( x_list[i-1], y_list[i-1])
    U_2 = Potential_energy( x_list[i], y_list[i])
    lnP = abs((U_2 - U_1))
    lnP_list[i-1] = lnP
```

```
HBox(children=(FloatProgress(value=0.0, max=1999999.0), HTML(value='')))
```

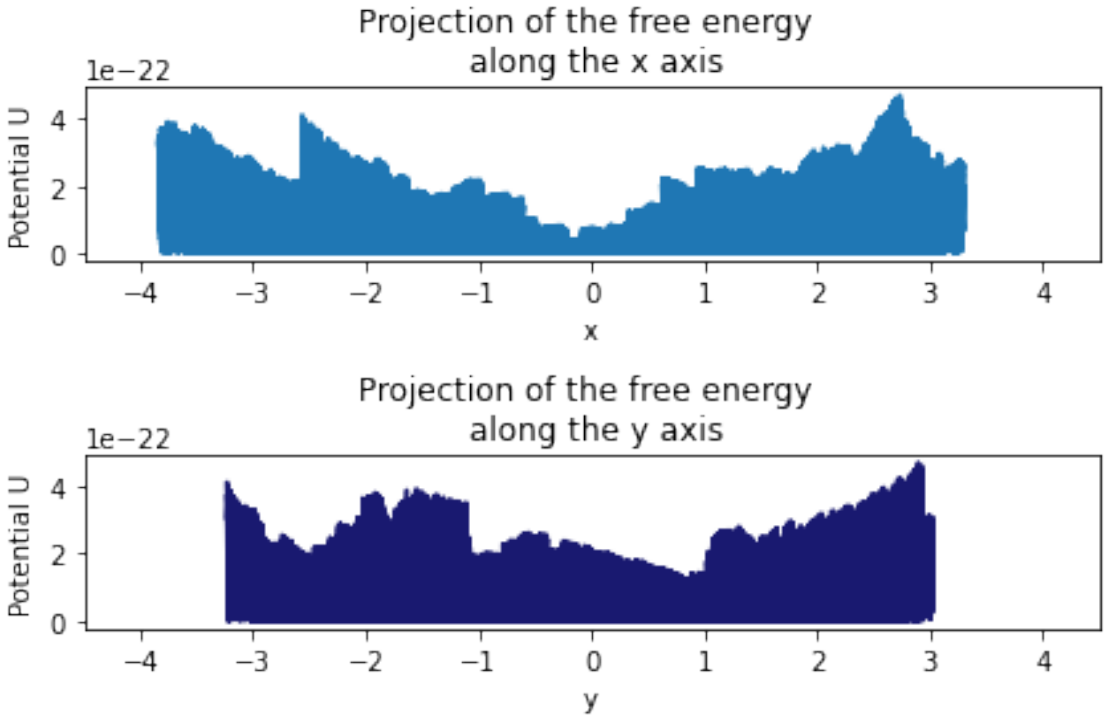
Plot

[11]:
```
plt.scatter(x_list[1:], y_list[1:], c = lnP_list[:],cmap=plt.
 ↪get_cmap('twilight'), s=3)
plt.scatter([2,-2], [1.5,-1.5], c = 'red', marker='x',label='Minima')

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-4.5,4.5)
plt.ylim(-4.5,4.5)

t = 'Free energy plot \n (with ' + str(Nbr_MC) + 'mooves)'
plt.title(t)

cbar = plt.colorbar(cm.ScalarMappable( cmap='twilight'))

lnP_ticks = np.linspace(min(lnP_list), max(lnP_list), 6)
lnP_ticks = [str(lnP_ticks[i])[0:3] + str(lnP_ticks[i])[-4:] for i in␣
 ↪range(len(lnP_ticks))]
cbar.ax.set_yticklabels(lnP_ticks)

cbar.set_label('Delta G', rotation=270)
```



The lighter areas are ones with the lowest free energy and correspondingly with the smallest potential gradient. We can see that the free energy is minimised around the area of minimum potential, which makes this plot quite similar to the potential plot above. But we also observe that along the conecting pathway between the minimas the free energy is also minimised. This is due to the fact that the simulation was run using the metropolis algorhthm, and the trajectory is more likely to follow a path with low gradient.

[53]:
```
plt.subplot(211)
plt.plot(x_list[0:-1], lnP_list)
t = 'Projection of the free energy \n along the x axis'
plt.title(t)
plt.xlim(-4.5, 4.5)
plt.xlabel('x')
plt.ylabel('Potential U')


plt.subplot(212)
t = 'Projection of the free energy \n along the y axis'
plt.title(t)
plt.xlim(-4.5, 4.5)
plt.xlabel('y')
plt.ylabel('Potential U')
plt.plot(y_list[0:-1], lnP_list, c = 'midnightblue')

plt.tight_layout()
```



When plotting the free energy along x and y we can't get any usefull inforation, wee see that there is a minimum in the first plot at values slightly lower then x = 0 but it is not shown clearly engough to be revelent.

1.5 Task IV: PCA

Means

[12]:
```
mean_x = np.mean(x_list)
mean_y = np.mean(y_list)
```

Variances

[13]:
```
var_x = np.var(x_list)
var_y = np.var(y_list)
```

Covariances and matrix σ_{ij}

$$\sigma_{ij} = \begin{pmatrix} var(x) & cov(x,y) \\ cov(y,x) & var(y) \end{pmatrix}$$

We want to maximise the varience as the more varience we have for a coordinate the more likely is it to describe the main characteristics from the potential, as most of the motion happends along new coordinate.

We also aim to minimilise the covarience. As the new set of coordinates should idealy be uncorrelated, which would mean cov(x,y)=0 and σ_{ij} is diagonal.

[14]:
```
sigma = np.cov(x_list, y_list)
print(' _ij =  \n',sigma)
```

```
 _ij =
 [[4.01749747 2.81880472]
 [2.81880472 2.31654261]]
```

Eigenvalues λ_k and eigenvectors $\vec{e}_{\lambda k}$

Diagonalising σ we have :

$$\sigma_{ij} = P\sigma_{diag}P^{-1}$$

with σ_{diag} being a diagonal matrix with the calculated eingen-values along the diagonal and P being the matrix with it's colomn vectors being the eingenvector of σ

$$\sigma_{diag} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = \begin{pmatrix} var(x') & cov(x',y') = 0 \\ cov(y',x') = 0 & var(y') \end{pmatrix}$$

$$P = \begin{pmatrix} e_{1X} & e_{2X} \\ e_{1Y} & e_{2Y} \end{pmatrix}$$

[67]:
```
eig_values, eig_vectors = np.linalg.eig(sigma)

eig_vec_1 = eig_vectors[:,0]
eig_val_1 = eig_values[0]
eig_vec_2 = eig_vectors[:,1]
eig_val_2 = eig_values[1]

print('Lambda1 = ', eig_val_1, 'with eigenvector e1: ', eig_vec_1)
print('Lambda2 = ', eig_val_2, 'with eigenvector e2: ',eig_vec_2)

n = np.sqrt((x_list[min_ind_2]-x_list[min_ind_1])**2 +␣
 ↪(y_list[min_ind_2]-y_list[min_ind_1])**2)
v1= [(x_list[min_ind_2]-x_list[min_ind_1])/
 ↪n,(y_list[min_ind_2]-y_list[min_ind_1])/n]
print('\n Unit vector connecting our two minimas : ',v1)
print()

sigma_diag = np.diag([eig_values[0],eig_values[1]])
print('Covarience matrix after diagonalisation : \n  _diag =  \n',sigma_diag)

P = np.matrix([[eig_vec_1[0],eig_vec_2[0]],[eig_vec_1[1],eig_vec_2[1]]])
print('\n P matrix : \n ', P)
```

```
Lambda1 =  6.1113318329016835 with eigenvector e1:  [0.80276223 0.59629925]
Lambda2 =  0.22270825116709414 with eigenvector e2:  [-0.59629925  0.80276223]

 Unit vector connecting our two minimas :  [0.8088054040909834,
0.58807637115703]

Covarience matrix after diagonalisation :
  _diag =
 [[6.11133183 0.        ]
 [0.         0.22270825]]

 P matrix :
  [[ 0.80276223 -0.59629925]
 [ 0.59629925  0.80276223]]
```

The eigenvectors match the data as expected, as we can see that our first vector (the one with higher eingenvalue ~6.11) has the same direction as the connecting axis between the two minimas and that our second vector is perpendicular to the first our as expected.

1.6 Task V: Interpretation

Angle θ between the eigenvector $\vec{e}_1$

[70]:
```
Theta = np.arctan(eig_vec_1[1]/eig_vec_1[0])
print('Theta ~', str(round(Theta/np.pi, 3)),' ')
```

```
Theta ~ 0.203
```

The angle θ found seems correct compared to the plot of the potential displayed in Task II.

Clockwise rotation matrix R

$$R(\theta) = \begin{pmatrix} Cos(\theta) & Sin(\theta) \\ -Sin(\theta) & Cos(\theta) \end{pmatrix}$$

$$R \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}$$

For x' and y' being the data in the rotated frame and x,y our initial data.

[71]:
```
R = np.array([[np.cos(Theta), np.sin(Theta)],[- np.sin(Theta), np.cos(Theta)]])
```

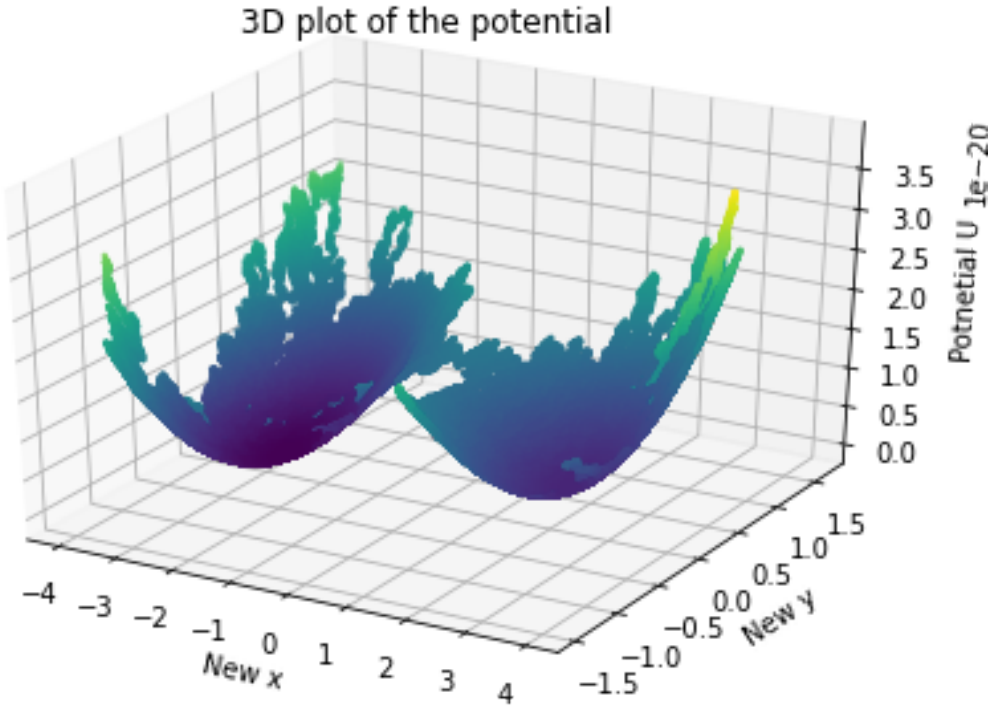
Calculate the data in the new frame

[72]:
```
x_rot_list = [R[0,0]*x_list[i] + R[0,1]*y_list[i] for i in trange(len(x_list))]
y_rot_list = [R[1,0]*x_list[i] + R[1,1]*y_list[i] for i in trange(len(x_list))]
```

```
HBox(children=(FloatProgress(value=0.0, max=2000001.0), HTML(value='')))
```

```
HBox(children=(FloatProgress(value=0.0, max=2000001.0), HTML(value='')))
```

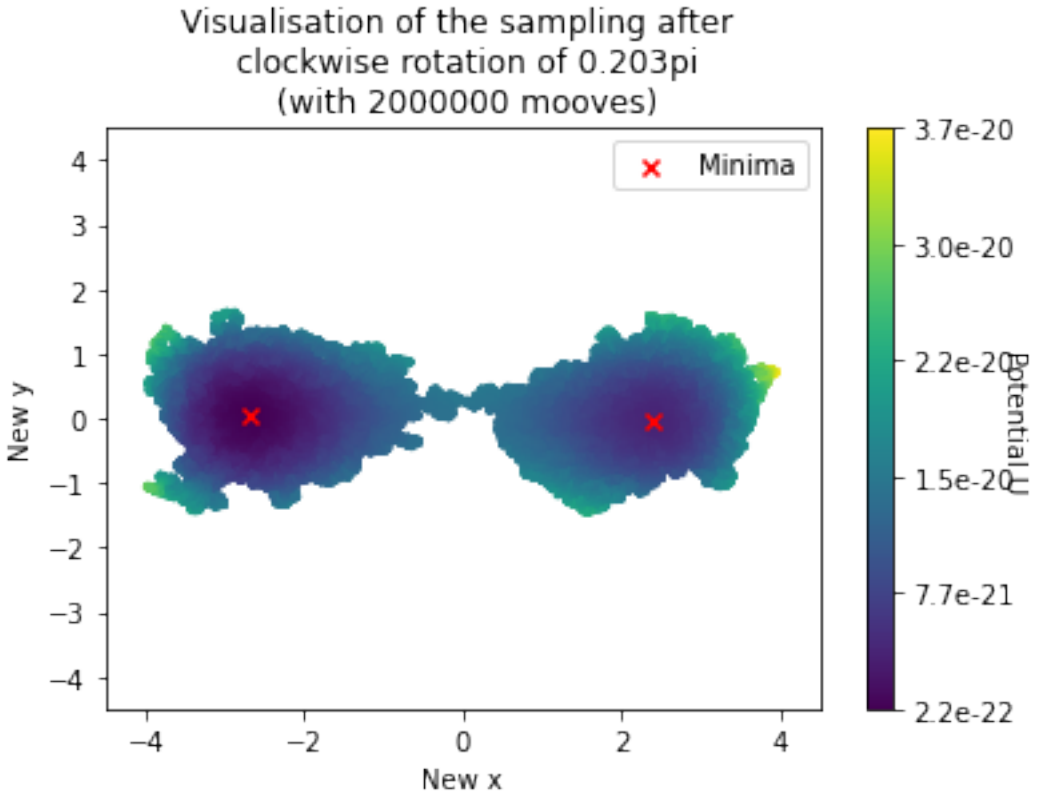
[75]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.scatter(x_rot_list, y_rot_list, U_list, c=U_list, marker='o', s=2)

plt.title('3D plot of the potential')

#ax.zaxis.set_major_locator(LinearLocator(10))
#ax.zaxis.set_major_formatter(U_ticks)
ax.set_xlabel('New x')
ax.set_ylabel('New y')
ax.set_zlabel('Potnetial U')

plt.tight_layout()
plt.show()
```



[73]:
```
plt.scatter(x_rot_list, y_rot_list, c = U_list, s=3)
plt.scatter([x_rot_list[min_ind_1],x_rot_list[min_ind_2]],␣
 ↪[y_rot_list[min_ind_1],y_rot_list[min_ind_2]], c = 'red',␣
 ↪marker='x',label='Minima')

plt.xlabel('New x')
plt.ylabel('New y')
plt.xlim(-4.5,4.5)
plt.ylim(-4.5,4.5)

t = 'Visualisation of the sampling after \n clockwise rotation of ' +␣
 ↪str(round(Theta/np.pi, 3)) + 'pi\n (with ' + str(Nbr_MC) + ' mooves)'
plt.title(t)
plt.legend()

cbar = plt.colorbar()

U_ticks = np.linspace(min(U_list), max(U_list), 6)
U_ticks = [str(U_ticks[i])[0:3] + str(U_ticks[i])[-4:] for i in␣
 ↪range(len(U_ticks))]
cbar.ax.set_yticklabels(U_ticks)

cbar.set_label('Potential U', rotation=270)
```

```
print('Minimum 1 at: (x,y) = (', str(x_rot_list[min_ind_1])[0:5],␣
 ↪str(y_rot_list[min_ind_1])[0:5], ') with U = ', U_min_1)
print()
print('Minimum 2 at: (x,y) = (', str(x_rot_list[min_ind_2])[0:5],␣
 ↪str(y_rot_list[min_ind_2])[0:5], ') with U = ', U_min_2)
```

```
Minimum 1 at: (x,y) = ( -2.69 0.027 ) with U =  2.2978051441872487e-22

Minimum 2 at: (x,y) = ( 2.399 -0.02 ) with U =  4.075809248912603e-21
```



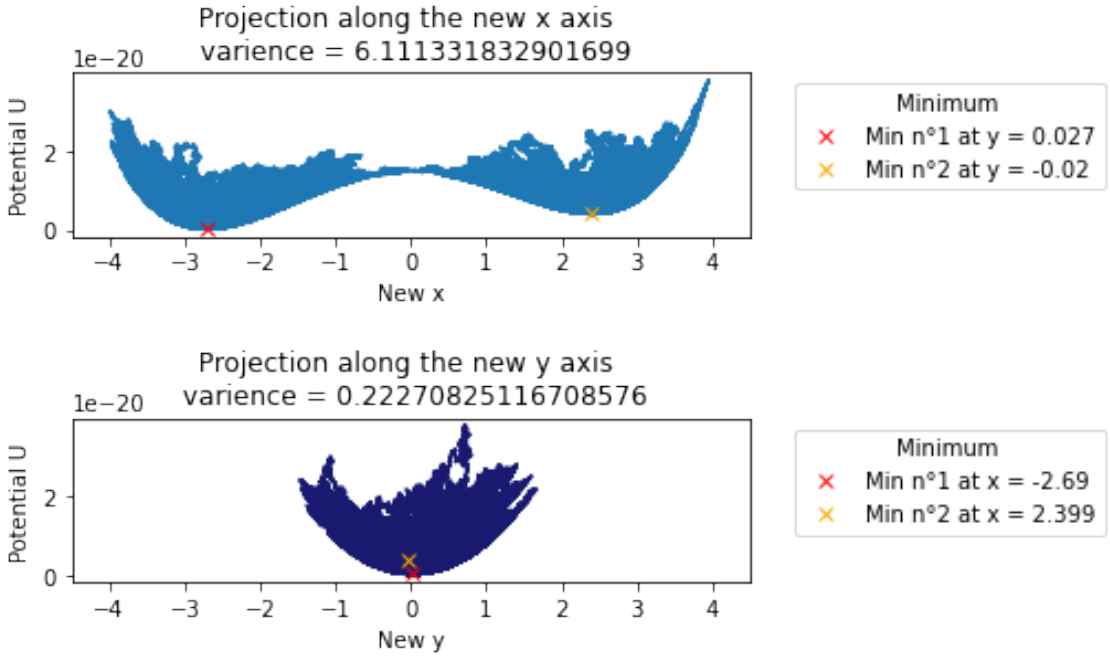
[79]:
```
#only used to find the varience and covarience between the new set of␣
 ↪coordinates.
sigma_new = np.cov(x_rot_list, y_rot_list)
print('Diagonalised sigma : \n ',sigma_diag, '\n \n Sigma matrix calculated␣
 ↪from the rotated data :\n ', sigma_new)

v1 = sigma[0,0]+ sigma[1,1]
print('\n Initial coordinates : \n ',str(sigma[0,0]/v1*100)[0:4],'% of the␣
 ↪varience of the total system is')
print('contained in x and ', str(sigma[1,1]/v1*100)[0:4],'% in y')
```

```
v2 = sigma_new[0,0]+ sigma_new[1,1]
print('\n New set of coordinates : \n ',str(sigma_new[0,0]/v2*100)[0:4],'% of␣
 ↪the varience of the total system is')
print('contained in the new x coordinate and ', str(sigma_new[1,1]/v2*100)[0:
 ↪4],'% in the new y')
```

```
Diagonalised sigma :
  [[6.11133183 0.        ]
 [0.         0.22270825]]

 Sigma matrix calculated from the rotated data :
  [[6.11133183e+00 1.15738032e-14]
 [1.15738032e-14 2.22708251e-01]]

 Initial coordinates :
  63.4 % of the varience of the total system is
contained in x and  36.5 % in y

 New set of coordinates :
  96.4 % of the varience of the total system is
contained in the new x coordinate and  3.51 % in the new y
```

[80]:
```
plt.subplot(211)
plt.plot(x_rot_list, U_list)

MIN1_X = 'Min n°1 at y = ' + str(y_rot_list[min_ind_1])[0:5]
MIN2_X = 'Min n°2 at y = ' + str(y_rot_list[min_ind_2])[0:5]
plt.plot([x_rot_list[min_ind_1]], [U_list[min_ind_1]],c = 'red', marker='x',␣
 ↪linestyle='None',label= MIN1_X )
plt.plot([x_rot_list[min_ind_2]], [U_list[min_ind_2]],c = 'orange', marker='x',␣
 ↪linestyle='None',label= MIN2_X )

t = 'Projection along the new x axis \n varience = ' + str(sigma_new[0,0])
plt.title(t)
plt.xlim(-4.5, 4.5)
plt.xlabel('New x')
plt.ylabel('Potential U')
plt.legend(title='Minimum ', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.subplot(212)
plt.plot(y_rot_list, U_list, c = 'midnightblue')

MIN1_Y = 'Min n°1 at x = ' + str(x_rot_list[min_ind_1])[0:5]
MIN2_Y = 'Min n°2 at x = ' + str(x_rot_list[min_ind_2])[0:5]
plt.plot([y_rot_list[min_ind_1]], [U_list[min_ind_1]],c = 'red', marker='x',␣
 ↪linestyle='None',label= MIN1_Y )
```

```
plt.plot([y_rot_list[min_ind_2]], [U_list[min_ind_2]],c = 'orange', marker='x',␣
↪linestyle='None',label= MIN2_Y )

t = 'Projection along the new y axis \n varience = ' + str(sigma_new[1,1])
plt.title(t)
plt.xlim(-4.5, 4.5)
plt.xlabel('New y')
plt.ylabel('Potential U')
plt.legend(title='Minimum ', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.subplots_adjust(bottom=0.1, right=0.8, top=0.9)
plt.show()
```

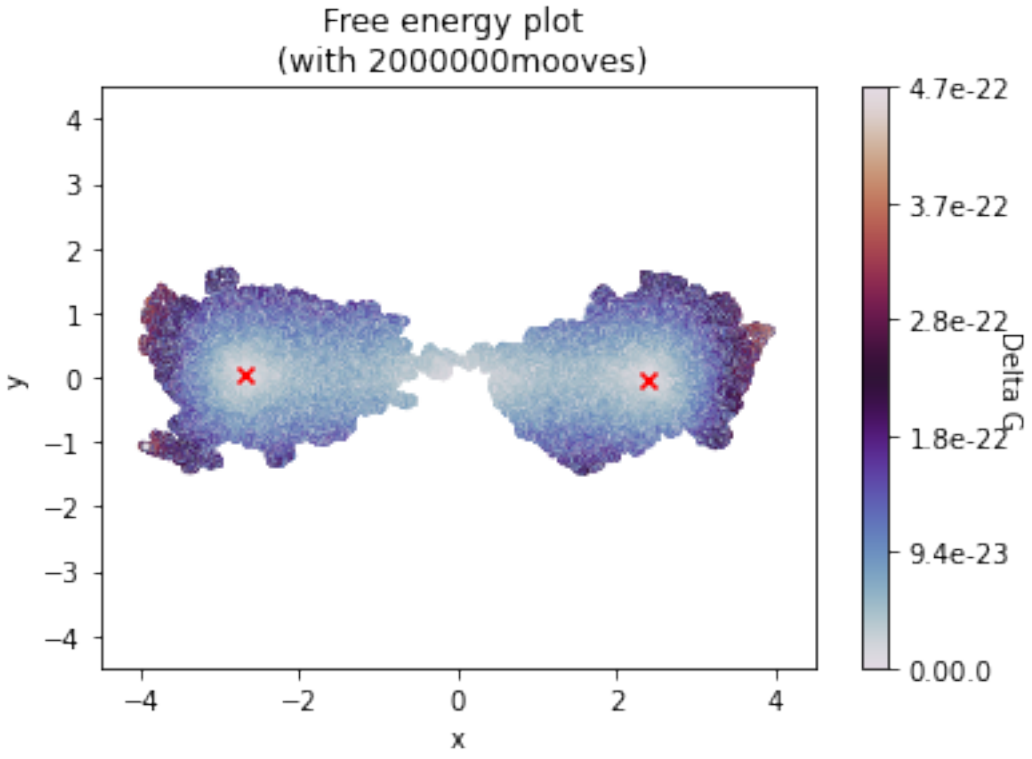


The main characteristics of the 2D energy surface are clearly visible when ploted along the new x axis, which is our eingenvector of the covarience matrix with the hightest eingenvalue (~6.11). We can distincly see the two minimas and the connecting path connecting them. As we saw the connecting path has a relativly low gradient.

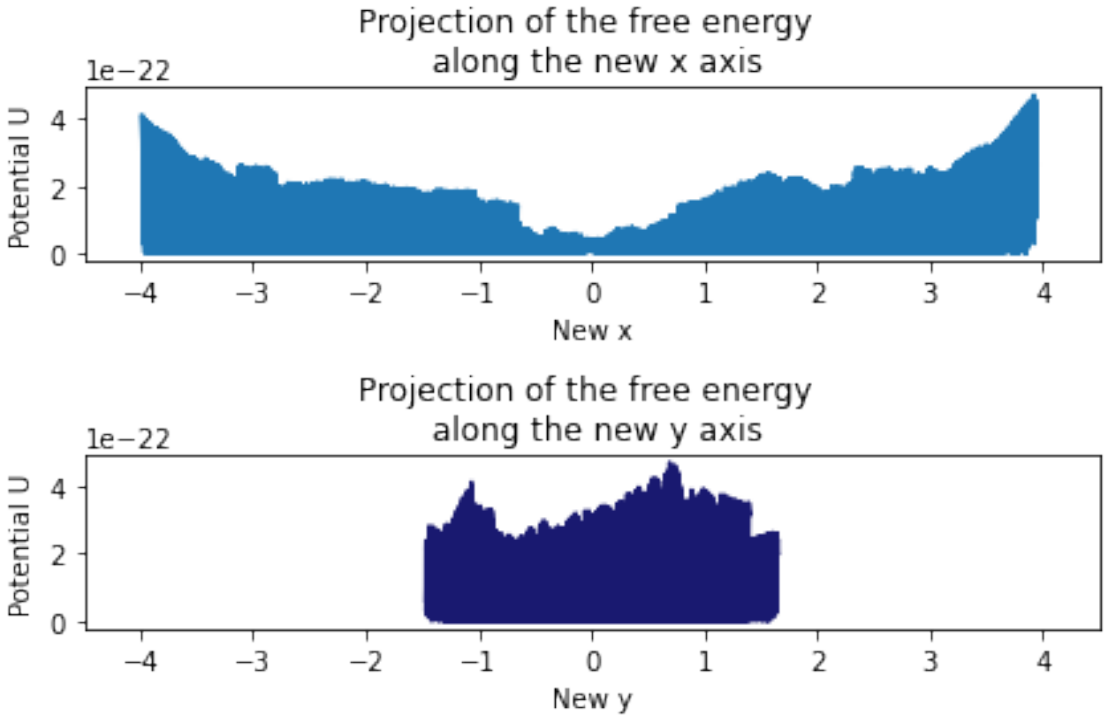
In the second plot, gathering only 3.51 % of the total varience of the system, we can nolonger dinstinguish the two minimas, it overall is gives off much less information about the shape of the potential and how would a system behave in it, as most of the motion happends along the first coordinate.

[81]:
```
plt.scatter(x_rot_list[1:], y_rot_list[1:], c = lnP_list[:],cmap=plt.
↪get_cmap('twilight'), s=3)
```

```
plt.scatter([x_rot_list[min_ind_1],x_rot_list[min_ind_2]],␣
 ↪[y_rot_list[min_ind_1],y_rot_list[min_ind_2]], c = 'red',␣
 ↪marker='x',label='Minima')

plt.xlabel('x')
plt.ylabel('y')
plt.xlim(-4.5,4.5)
plt.ylim(-4.5,4.5)

t = 'Free energy plot \n (with ' + str(Nbr_MC) + 'mooves)'
plt.title(t)

cbar = plt.colorbar(cm.ScalarMappable( cmap='twilight'))

lnP_ticks = np.linspace(min(lnP_list), max(lnP_list), 6)
lnP_ticks = [str(lnP_ticks[i])[0:3] + str(lnP_ticks[i])[-4:] for i in␣
 ↪range(len(lnP_ticks))]
cbar.ax.set_yticklabels(lnP_ticks)

cbar.set_label('Delta G', rotation=270)
```



[82]:
```
plt.subplot(211)
plt.plot(x_rot_list[0:-1], lnP_list)
t = 'Projection of the free energy \n along the new x axis'
plt.title(t)
plt.xlim(-4.5, 4.5)
plt.xlabel('New x')
plt.ylabel('Potential U')


plt.subplot(212)
t = 'Projection of the free energy \n along the new y axis'
plt.title(t)
plt.xlim(-4.5, 4.5)
plt.xlabel('New y')
plt.ylabel('Potential U')
plt.plot(y_rot_list[0:-1], lnP_list, c = 'midnightblue')

plt.tight_layout()
```



We can see that the free energy is lower around new x = 0, with is the connection between the minimas. We can clearly see it here whereas we can barely see it is we plot it along x and y, the initial set of coordinates.

1.7 Results and discussion

From the projection of the potential along the new set of coordinate we can observe the difference of motion along the the principal component 'New x' and the second one 'New y'. By rotating the data we manage to go from 63.4 % of the varience in the along the first dimention to 96.4 %, which leave us with a systemthat can be studied

The display of the Free energy doesn't seem to be ideal. Apart form the fact that it quite clearly indicates that the free energy is low along the path connecting the two minimas, it does not give much information about the system.

1.8 Conclusion

Even with a small 2D system we saw how principal componant analysis can be used to construct a new set of uncorrelated coordinates that can be ordered by their varience. In our simulation, by calculating those new coordinates we observed, as expected, that the first one gathered 96.4% of the varience of the system, and displayed the main characteristcs of the energy surface studied.
We also saw that the potential barrier separating the two minimas had a low free energy along the path which is consistant with the fact that, using a Metropolis algorythm we are more likely to go over a potential if it has a low gradiend.

This method can be used with much larger systems, with N dimentions and allows to select only a few (1 to 5) dimentions which still allows us to study the system in a revelent way but in much lower dimentions.