(a)

At first declared initial probabilities and transition matrix

```
# initial state probabilities
PI= [0.5, 0.1, 0.3,0.1]
```

```
# transition matrix A
A=np.array([ [0.2, 0.2, 0.3,0.3],
    [0.1, 0.3, 0.2,0.4],
    [0.1, 0.2, 0.3,0.4],
    [0.1, 0.1, 0.1,0.7]
  ])
```

Created np array of 200 row of 500 column with 0,1,2,3 values as the states are 4. The following figure shows values of first row.

```
# generate 200 sequence of 500 states
seq2 = np.random.randint(4, size=(200, 500))
seq2[0]
```

```
array([2, 3, 0, 2, 2, 3, 0, 0, 2, 1, 2, 2, 2, 2, 3, 0, 3, 3, 3, 2, 1, 0,
       1, 3, 3, 1, 1, 1, 3, 3, 0, 0, 3, 1, 1, 0, 3, 0, 0, 2, 2, 2, 1, 3,
       3, 3, 3, 2, 1, 1, 2, 1, 2, 3, 2, 3, 3, 0, 2, 0, 2, 2, 0, 0, 2, 1,
       3, 0, 3, 1, 1, 1, 0, 1, 0, 1, 3, 3, 2, 3, 2, 3, 0, 3, 2, 2, 1, 0,
       3, 1, 3, 3, 1, 1, 1, 1, 1, 3, 1, 0, 2, 1, 1, 3, 1, 1, 1, 3, 1, 2,
       3, 2, 3, 1, 2, 3, 0, 1, 3, 0, 3, 0, 1, 2, 0, 3, 1, 0, 3, 3, 3, 0,
       0, 0, 2, 0, 0, 0, 2, 0, 3, 0, 3, 3, 3, 2, 2, 2, 0, 3, 2, 2, 0, 2,
       0, 1, 2, 1, 0, 3, 2, 0, 3, 3, 1, 0, 3, 2, 2, 1, 3, 0, 2, 3, 3, 1,
       2, 2, 0, 2, 0, 2, 1, 2, 0, 0, 1, 2, 2, 1, 2, 2, 0, 2, 2, 1, 1, 3,
       0, 2, 2, 3, 2, 0, 3, 0, 3, 3, 1, 0, 2, 2, 0, 2, 2, 0, 3, 0, 3, 2,
       2, 2, 1, 3, 1, 1, 0, 1, 0, 0, 1, 3, 3, 3, 3, 3, 1, 1, 2, 3, 1, 2,
       3, 0, 2, 1, 0, 0, 0, 2, 1, 0, 3, 0, 0, 2, 2, 1, 3, 2, 0, 1, 0, 0,
       2, 1, 3, 3, 2, 2, 1, 3, 3, 3, 0, 3, 0, 1, 0, 1, 3, 3, 1, 2, 1, 2,
       0, 0, 0, 3, 0, 2, 0, 1, 1, 3, 1, 2, 0, 3, 0, 0, 2, 1, 1, 0, 3, 1,
       3, 1, 3, 2, 3, 2, 2, 3, 2, 0, 2, 1, 3, 0, 3, 1, 1, 1, 2, 2, 3, 3,
       0, 0, 3, 2, 1, 3, 0, 2, 3, 3, 2, 3, 2, 1, 2, 2, 2, 3, 3, 2, 3, 0,
       0, 3, 3, 2, 1, 3, 0, 2, 3, 0, 0, 1, 2, 2, 1, 1, 2, 2, 3, 3, 1, 3,
       0, 3, 3, 0, 1, 0, 3, 1, 3, 0, 0, 3, 2, 2, 0, 0, 2, 2, 3, 3, 2, 3,
       3, 1, 3, 0, 3, 3, 2, 0, 3, 3, 1, 3, 3, 1, 2, 3, 1, 0, 2, 0, 0, 1,
       3, 1, 1, 1, 1, 2, 3, 0, 0, 3, 0, 3, 0, 1, 0, 3, 3, 3, 3, 3, 2, 2,
       0, 3, 0, 3, 3, 2, 2, 1, 2, 1, 1, 1, 1, 1, 0, 2, 1, 3, 2, 2, 1, 0,
       1, 0, 3, 2, 3, 0, 0, 3, 0, 3, 2, 3, 2, 3, 0, 3, 0, 3, 3, 1, 0, 1,
       0, 1, 2, 3, 0, 0, 3, 0, 0, 1, 0, 2, 2, 3, 0, 3])
```

After create matrix, values are converted to string and joint all the values of each row as big string which is 500 sequence of 4 states they are 0,1,2,3.
The figure shows first value of seq.

```python
seq1 = [",".join(item) for item in seq2.astype(str)]
```

```python
seq1[0]
```

```
'2,3,0,2,2,3,0,0,2,1,2,2,2,2,3,0,3,3,3,2,1,0,1,3,3,1,1,1,3,3,0,0,3,1,1,0,3,0,0,2,2,2,1,3,3,3,3,2,1,1,2,1,2,3,2,3,3,0,2,0,2,2,0,
0,2,1,3,0,3,1,1,1,0,1,0,1,3,3,2,3,2,3,0,3,2,2,1,0,3,1,3,3,1,1,1,1,1,3,1,0,2,1,1,3,1,1,1,3,1,2,3,2,3,1,2,3,0,1,3,0,3,0,1,2,0,3,
1,0,3,3,3,0,0,0,2,0,0,0,2,0,3,0,3,3,3,2,2,2,0,3,2,2,0,2,0,1,2,1,0,3,2,0,3,3,1,0,3,2,2,1,3,0,2,3,3,1,2,2,0,2,0,2,1,2,0,0,1,2,2,
1,2,2,0,2,2,1,1,3,0,2,2,3,2,0,3,0,3,3,1,0,2,2,0,2,2,0,3,0,3,2,2,2,1,3,1,1,0,1,0,0,1,3,3,3,3,3,1,1,2,3,1,2,3,0,2,1,0,0,0,2,1,0,
3,0,0,2,2,1,3,2,0,1,0,0,2,1,3,3,2,2,1,3,3,3,0,3,0,1,0,1,3,3,1,2,1,2,0,0,0,3,0,2,0,1,1,3,1,2,0,3,0,0,2,1,1,0,3,1,3,1,3,2,3,2,2,
3,2,0,2,1,3,0,3,1,1,1,2,2,3,3,0,0,3,2,1,3,0,2,3,3,2,3,2,1,2,2,2,3,3,2,3,0,0,3,3,2,1,3,0,2,3,0,0,1,2,2,1,1,2,2,3,3,1,3,0,3,3,0,
1,0,3,1,3,0,0,3,2,2,0,0,2,2,3,3,2,3,3,1,3,0,3,3,2,0,3,3,1,3,3,1,2,3,1,0,2,0,0,1,3,1,1,1,1,2,3,0,0,3,0,3,0,1,0,3,3,3,3,3,2,2,0,
3,0,3,3,2,2,1,2,1,1,1,1,1,0,2,1,3,2,2,1,0,1,0,3,2,3,0,0,3,0,3,2,3,2,3,0,3,0,3,3,1,0,1,0,1,2,3,0,0,3,0,0,1,0,2,2,3,0,3'
```

```python
seq = [s.replace(",", "") for s in seq1]
```

```python
# States 0,1,2,3
# seq contains 200 sequences of 500 states
seq[0]
```

```
'230223002122223033321013311133003110300222133332112123233020220021303111010133232303221031331111131021131113123231230130301203
1033300020002030333222032202012103203310322130233122020212001221220221130223203033102202203032221311010013333311231230210002103
0022132010021332213330301013312120003020113120300211031313232232021303111223300321302332321222332300332130230012211223313033010
3130032200223323313033203313312310200131111230030301033333220303322121111102132210103230030323230303310101230030010223 03'
```

(b)

For PI calculation we created States list and put all values to states. Then calculate distribution probabilities of each states ‘0’,’1’,’2’,’3’ by their frequency and get PI values divide frequency by total number of state occurance.

PI Calculation from Sequence (200 sequence of 500 states)

```python
# Derive States 0,1,2,3 from seq
```

```python
States=[]
for w in seq:
 for c in w:
   if c not in States:
    States.append(c)
```

```python
States
```

```
['2', '3', '0', '1']
```

```python
PI_Calculation= np.zeros(4, dtype="float64")
```

```python
for w in seq:
 PI_Calculation[States.index(w[0])] +=1

PI_Calculation= PI_Calculation/sum(PI_Calculation)
```

```python
PI_Calculation
```

```
array([0.295, 0.22 , 0.23 , 0.255])
```

Secondly we calculated transition matrix. First initialize numpy zero 4 by 4 matrix. Then took seq made process it like pairs by each single character of string. For instance if the string is ‘012’ it will return (0,1),(1,2). Then count the frequency of each states (0,1,2,3) and put it to the matrix. Codes are below

Transition Matrix Calculation

```
A_Calculation= np.zeros((4, 4))
```

```
for word in seq:
  W= list(word)
  bi=list(zip(W,W[1:]))
  for p in bi:
   i=States.index(p[0])
   j=States.index(p[1])
   A_Calculation[i,j]+= 1
```

```
A_Calculation
```

```
array([[6115., 6251., 6259., 6208.],
       [6161., 6219., 6234., 6305.],
       [6249., 6235., 6388., 6131.],
       [6304., 6223., 6122., 6396.]])
```

Then we add each row of the A_Calculation matrix and divide each value of the array with total sum of that row. Here B is the new matrix of transition probability.

```
rowSum=np.sum(A_Calculation,1)
```

```
rowSum
```

```
array([24833., 24919., 25003., 25045.])
```

```
B=(A_Calculation.T/rowSum).T
```

```
B
```

```
array([[0.24624492, 0.2517215 , 0.25204365, 0.24998993],
       [0.24724106, 0.2495686 , 0.25017055, 0.25301978],
       [0.24993001, 0.24937008, 0.25548934, 0.24521057],
       [0.25170693, 0.24847275, 0.24444001, 0.25538032]])
```

```
rowSumB=np.sum(B,1)
```

```
rowSumB
```

```
array([1., 1., 1., 1.])
```

After round the value of B we get Transition matrix

```
A_Calculation=B
A_transition_calculation = np.round(A_Calculation,2)
```

```
A_transition_calculation
```

```
array([[0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.26, 0.25],
       [0.25, 0.25, 0.24, 0.26]])
```

Lastly we just subtract derived initial probability and transition matrix from defined matrixes.

Comarision of Initial and transition matrix

```
np.asarray(PI)
```

```
array([0.5, 0.1, 0.3, 0.1])
```

```
PI_Difference = PI - PI_Calculation
```

```
A_Difference = A - A_transition_calculation
```

```
# Diffrence of the values
```

```
PI_Difference
```

```
array([ 0.205, -0.12 ,  0.07 , -0.155])
```

```
A_Difference
```

```
array([[-0.05, -0.05,  0.05,  0.05],
       [-0.15,  0.05, -0.05,  0.15],
       [-0.15, -0.05,  0.04,  0.15],
       [-0.15, -0.15, -0.14,  0.44]])
```

(c)

We define the packages and generate GausianHMM viterbi algorithms with full covariance matrix and 500 iteration for generate 500 states. We initialised initial probability and transition matrix for the model. Model creates emission matrix by “Gaussian emission” method.

```
from hmmlearn import hmm
np.random.seed(42)
```

```
# GaussianHMM has used as it provides a very compact representation of the data.
# GaussianHMM genarate viterbi algorithms
# Hidden Markov Model with Gaussian emissions has created.
# We initialize 4 states as n_components.
# covariance type is full where each state uses a full covariance matrix.

# Initial Probability distribution
startprob_ = np.array([0.5, 0.1, 0.3,0.1])

# Transition probability matrix
transmat_ = np.array([ [0.2, 0.2, 0.3,0.3],
    [0.1, 0.3, 0.2,0.4],
    [0.1, 0.2, 0.3,0.4],
    [0.1, 0.1, 0.1,0.7]
  ])
means_ = np.array([[0.0, 0.0], [3.0, 3.0], [5.0, 10.0],[1.0,1.0]])
covars_ = np.tile(np.identity(2), (4, 1,1))

# n_iter=500 will genarate 500 states
model = hmm.GaussianHMM(n_components=4,n_iter=500, covariance_type="full")

# Instead of fitting it from the data, we directly set the estimated
# parameters, the means and covariance of the components
model.startprob_ = startprob_
model.transmat_ = transmat_
model.means_ = means_
model.covars_ = covars_
```

Generated 200 samples of data. Fit the data during remodel and predict the states. Have shown monitor data and sample data

```
# Generate samples
X, Z = model.sample(200)
print(len(X))
```

```
200
```

Fitting HMMLearn library for sample 200 sequences of 500 states.

```
remodel = hmm.GaussianHMM(n_components=4, covariance_type="full", n_iter=500)
remodel.fit(X)
```

```
GaussianHMM(covariance_type='full', n_components=4, n_iter=500)
```

```
Z2 = remodel.predict(X) # hidden state sequence
```

```
remodel.monitor_
```

```
ConvergenceMonitor(
    history=[-864.2959544566986, -774.3175282471484, -747.3352395349646, -734.4979796072162, -729.6010211705734, -728.514854148
571, -727.7220146221015, -727.0661761707763, -726.49061909619, -725.9767903571866, -725.524558187613, -725.1356780946946, -724.
8052181462383, -724.5237783683549, -724.2820450049616, -724.0727869296985, -723.8907513356304, -723.7318929429864, -723.5927090
589923, -723.4698969074861, -723.3602909160616, -723.2609542386685, -723.1693063194936, -723.0832146755513, -723.0010270703436,
-722.9215507319121, -722.8439966099924, -722.7679061730341, -722.6930736207419, -722.6194719871543, -722.5471885940045, -722.47
6373212402, -722.407200377143, -722.3398452935733, -722.2744709328643, -722.2112227184135, -722.1502270187995, -722.09159046193
4, -722.0353984777187, -721.9817128916999, -721.9305693481265, -721.8819756319946, -721.8359116849988, -721.7923315560056, -72
1.751166993175, -721.7123320602636, -721.6757280853508, -721.6412483635152, -721.6087822352662, -721.5782183656046, -721.549447
2058937, -721.5223627188942, -721.4968634932814, -721.4728533836781, -721.4502418008012, -721.4289437557194, -721.408879739615
3, -721.3899754998598, -721.3721617560564, -721.355373886515, -721.3395516055376, -721.3246386448268, -721.3105824472264, -721.
2973338775697, -721.2848469530446, -721.2730785940045, -721.2619883951173, -721.2515384162069, -721.2416929917255],
    iter=69,
    n_iter=500,
    tol=0.01,
    verbose=False,
)
```

```
remodel.sample(200) # observation sequence
```

```
        [-4.95456587e-01,  4.00640670e-01],
        [ 1.78658344e+00,  7.59575974e-01],
        [-1.09456500e+00,  2.51875825e-01],
        [ 4.08462181e+00,  9.83840521e+00],
        [ 5.04975416e+00,  1.06152420e+01],
        [ 2.06580459e+00, -1.47961614e-01],
        [ 2.16120084e+00,  5.92038093e-01],
        [ 1.31073495e+00,  3.14282535e+00],
        [ 5.45851081e+00,  9.61272943e+00],
        [ 1.68748794e-02,  2.12577696e-01]]),
 array([3, 1, 1, 2, 1, 2, 2, 3, 1, 3, 0, 3, 1, 1, 3, 0, 3, 0, 3, 0, 3, 0,
        2, 3, 0, 3, 0, 2, 2, 3, 0, 2, 2, 1, 3, 0, 1, 3, 0, 3, 0, 2, 2, 2,
        2, 2, 2, 3, 0, 3, 1, 2, 3, 1, 1, 2, 3, 1, 2, 1, 2, 1, 2, 2, 1, 3,
        0, 2, 3, 0, 3, 0, 2, 3, 0, 3, 0, 3, 0, 2, 2, 3, 0, 3, 0, 3, 0, 2,
        3, 0, 0, 3, 0, 0, 2, 3, 0, 2, 1, 1, 2, 1, 3, 0, 3, 0, 3, 0, 3, 1,
```

Estimate PI and transition matrix and print them

Estimatation of PI

```
print("HMM Start Probablity PI: ", remodel.startprob_)
```

```
HMM Start Probablity PI:  [1.61335323e-80 0.00000000e+00 8.00748124e-69 1.00000000e+00]
```

Estimatation Of Transition Matrix A

```
print("Transition matrix A", remodel.transmat_)
```

```
Transition matrix A [[1.20255632e-01 5.67768756e-02 3.42247624e-01 4.80719869e-01]
 [2.50931486e-05 2.10365471e-01 4.91627785e-01 2.97981651e-01]
 [1.40688476e-02 2.09322331e-01 3.79523620e-01 3.97085201e-01]
 [8.02980331e-01 1.78072096e-01 1.16373751e-02 7.31019768e-03]]
```

Calculated the difference from defined PI and Transition matrix.

Difference of matrix (Transition and Initial)

```
Transition_Matrix_Difference = remodel.transmat_ - A
```

```
Transition_Matrix_Difference
```

```
array([[-0.07974437, -0.14322312,  0.04224762,  0.18071987],
       [-0.09997491, -0.08963453,  0.29162778, -0.10201835],
       [-0.08593115,  0.00932233,  0.07952362, -0.0029148 ],
       [ 0.70298033,  0.0780721 , -0.08836262, -0.6926898 ]])
```

```
Initial_probability_Difference = remodel.startprob_ - PI
```

```
Initial_probability_Difference
```

```
array([-0.5, -0.1, -0.3,  0.9])
```