19CS10071 Assgn 1

Q9) (i) Random Initialisation:

Number of iterations: 32

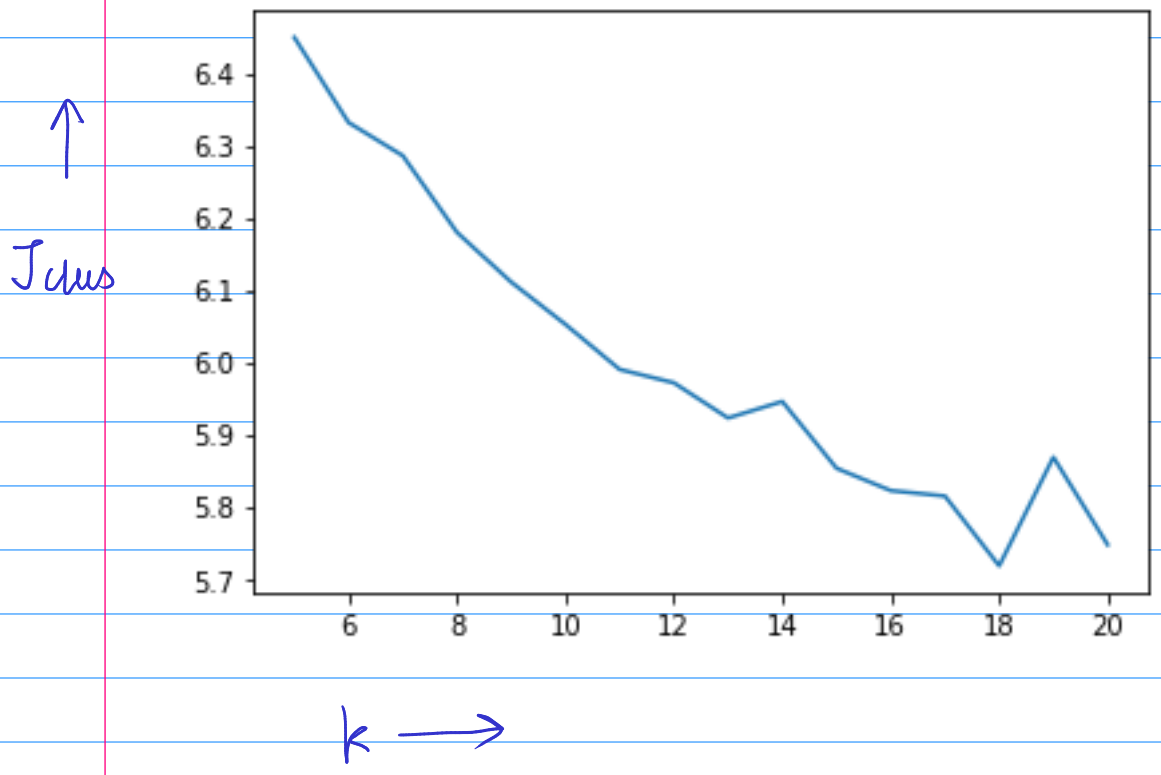
Plot of cluster Representatives (with predicted value based on max freq. value of each cluster)

[generated using matplotlib]

b) Accuracy of cluster assignment: 0.66

[on test sample of 50 images] = 66%

c) Plot of optimal J_{clus} vs k:

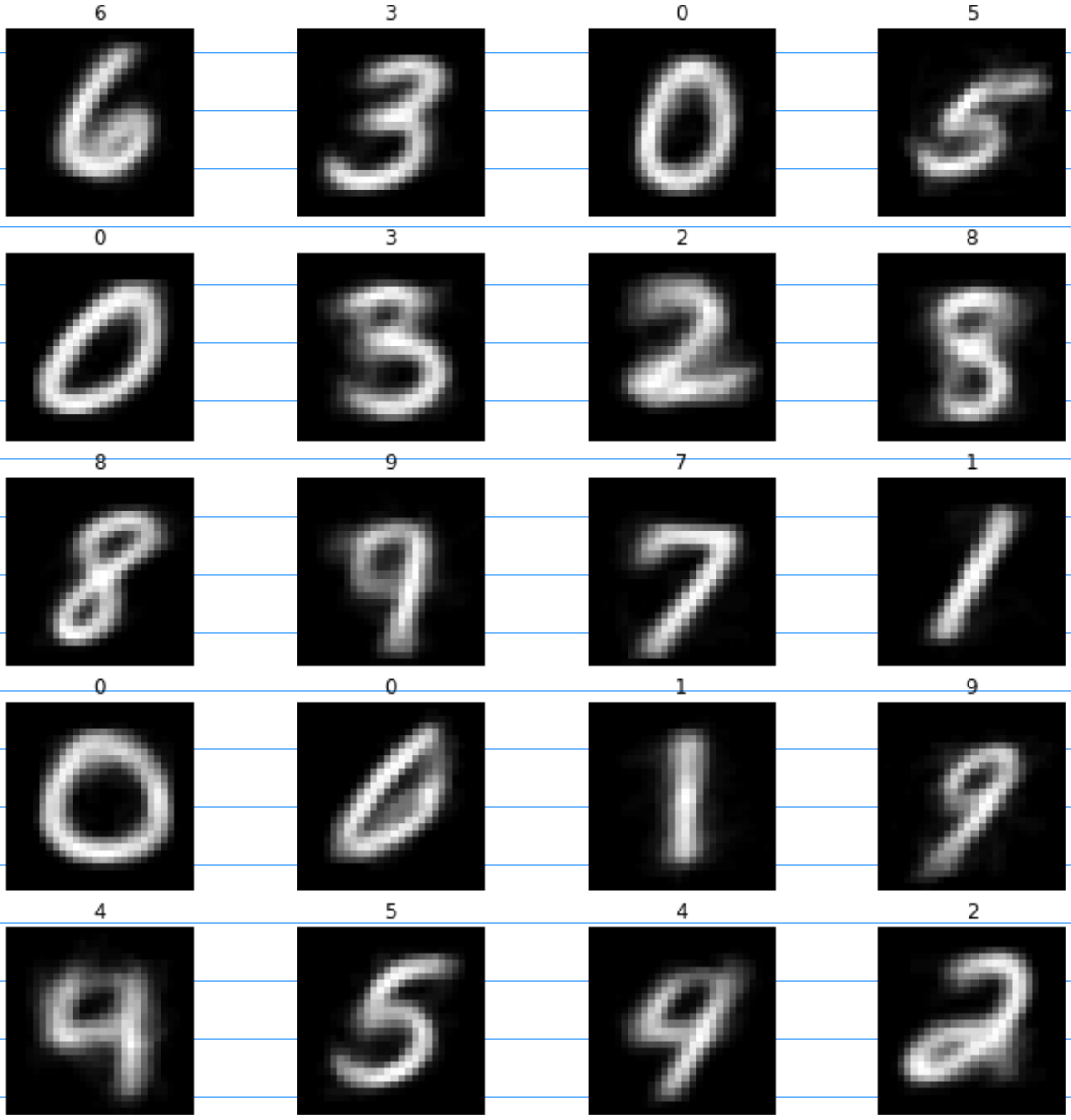


From here we find optimal k is around 18.

(ii) Cluster representatives chosen from given dataset:

a) Number of iterations: 23

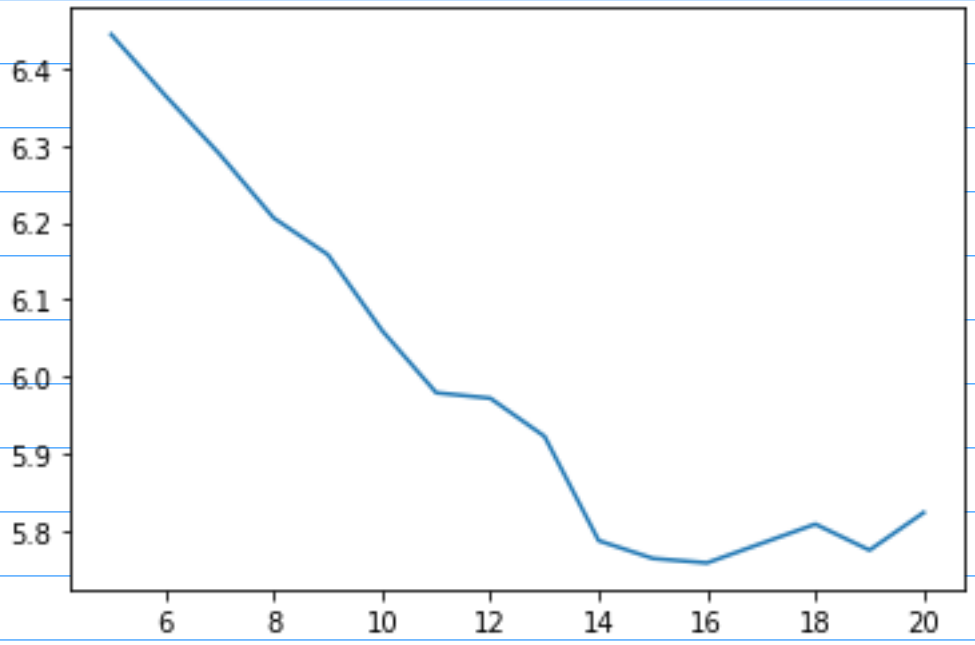
Plot of cluster representatives:



b) Accuracy on Test Data

$= 0.7 = 70\%$

c) Optimal J_{us} vs k plot:



J_{us} ↑

k →

So we find optimal k is around 16

Yes choice of initial k does have an effect on clustering algorithm. For random initialization, we get more iterations and lesser accuracy. For initializing from examples it converges faster and has more accuracy.

mnist_classifier

September 14, 2021

[1]:
```
import numpy as np
```

[2]:
```
from keras.datasets import mnist
```

[3]:
```
#load data
(train_X,train_y),(val_X,val_y) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

[4]:
```
train_X = np.asarray(train_X)
```

[5]:
```
train_X.shape
```

[5]:
```
(60000, 28, 28)
```

[6]:
```
train_y = np.asarray(train_y)
val_X = np.asarray(val_X)
val_y = np.asarray(val_y)
```

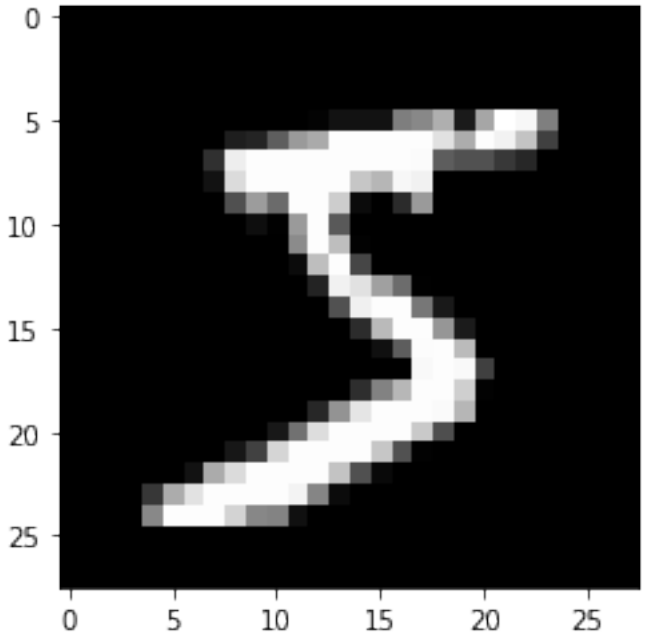
[7]:
```
val_X.shape
```

[7]:
```
(10000, 28, 28)
```

[8]:
```
import matplotlib.pyplot as plt

plt.imshow(train_X[0],cmap="gray")
print(train_y[0])
```

```
5
```



[9]:
```
train_X = train_X.reshape((60000,784))
val_X = val_X.reshape((10000,784))
```

[10]:
```
train_X = train_X.astype(float)/255.0
```

[11]:
```
val_X = val_X.astype(float)/255.0
```

[12]:
```
X = []
y = []
frq = [100]*10
for i in range(60000):
  if(frq[train_y[i]]>0):
    frq[train_y[i]]-=1
    X.append(train_X[i])
    y.append(train_y[i])
X = np.array(X)
y = np.array(y)
print(frq)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[13]:
```
y.shape
```

[13]: (1000,)

[14]:
```
class KMeans:
  def __init__(self,nCls,Xt,yt,**kwargs):
    self.k = nCls #number of clusters
    self.X = Xt
    self.y = yt
    self.n = len(self.X)
    self.c = np.zeros(self.X.shape[0],dtype=int)
    self.Z = np.random.rand(nCls,self.X.shape[1])#initialize with random values
    if('seed' in kwargs): self.Z = kwargs['seed']
    #print(self.Z)
    #print(self.X)
    self.Jclus = 0
    self.frq  = np.zeros((nCls,1))
    self.reassignCluster()
    self.calcJclus()
    self.refDict = {}
    self.iterations = 0
    #print(self.frq)
  def calcJclus(self):
    self.Jclus = 0
    for i in range(self.n):
      self.Jclus+=np.linalg.norm(self.X[i]-self.Z[self.c[i]])
    self.Jclus/=float(self.n)
  def reassignClusterRepr(self):
    self.Z = np.zeros((self.k,self.X.shape[1]))
    for i in range(self.n):
      self.Z[self.c[i]]=self.Z[self.c[i]]+self.X[i]
    for i in range(self.k):
      if(self.frq[i]!=0):  self.Z[i]/=self.frq[i]

  def reassignCluster(self):
    #print('1 '+self.frq.shape)
    self.frq = np.zeros((self.k,1))
    #print('2 '+self.frq.shape)
    for i in range(self.n):
      self.c[i] = np.argmin(np.linalg.norm(self.Z-self.X[i],axis=1))
      self.frq[self.c[i]]+=1
  def getClusterRepr(self):
    return self.Z
  def getRefDict(self):
    #This code maps the cluster ids to the predicted numbers
    #This is based on the approach that the one with majority freq in a cluster␣
↪is chosen
    return self.refDict #computyed in fir func
  def clusterId(self,img):
    #img must be in form (28*28) normalised to [0,1]
    return np.argmin(np.linalg.norm(self.Z-img,axis=1))
```

```
  def predict(self,img):
    return self.refDict[self.clusterId(img)]
  def fitData(self):
    oldJclus = 0.0
    while np.abs(oldJclus-self.Jclus)>1e-6:
      self.reassignCluster()
      self.reassignClusterRepr()
      oldJclus = self.Jclus
      self.calcJclus()
      self.iterations+=1
    #construct ref dict
    freq = np.zeros((self.k,10),dtype=int)
    for i in range(self.n):
      freq[self.c[i]][self.y[i]]+=1
    self.refDict = {}
    for i in range(self.k):
      self.refDict[i] = np.argmax(freq[i])
```

[15]:
```
def getSeed(nClus):
  '''
  initZ = [None]*nClus
  for i in range(len(X)):
    for j in range(int((nClus+9)/10)+1):
      if(j*10+y[i]<len(initZ)):
        if(initZ[j*10+y[i]] is None):
          initZ[y[i]+j*10] = X[i]
          '''
  choices = np.random.choice(1000,size=nClus)
  Zinit = X[choices]
  return Zinit
Z = getSeed(20)
print(Z.shape)
```

```
(20, 784)
```

[50]:
```
nClust = 20
model = KMeans(nClust,X,y,seed=getSeed(nClust))
model.fitData()
```

[51]:
```
model.iterations
```

[51]:
```
23
```

[52]:
```
refTable = model.getRefDict()
refTable
```

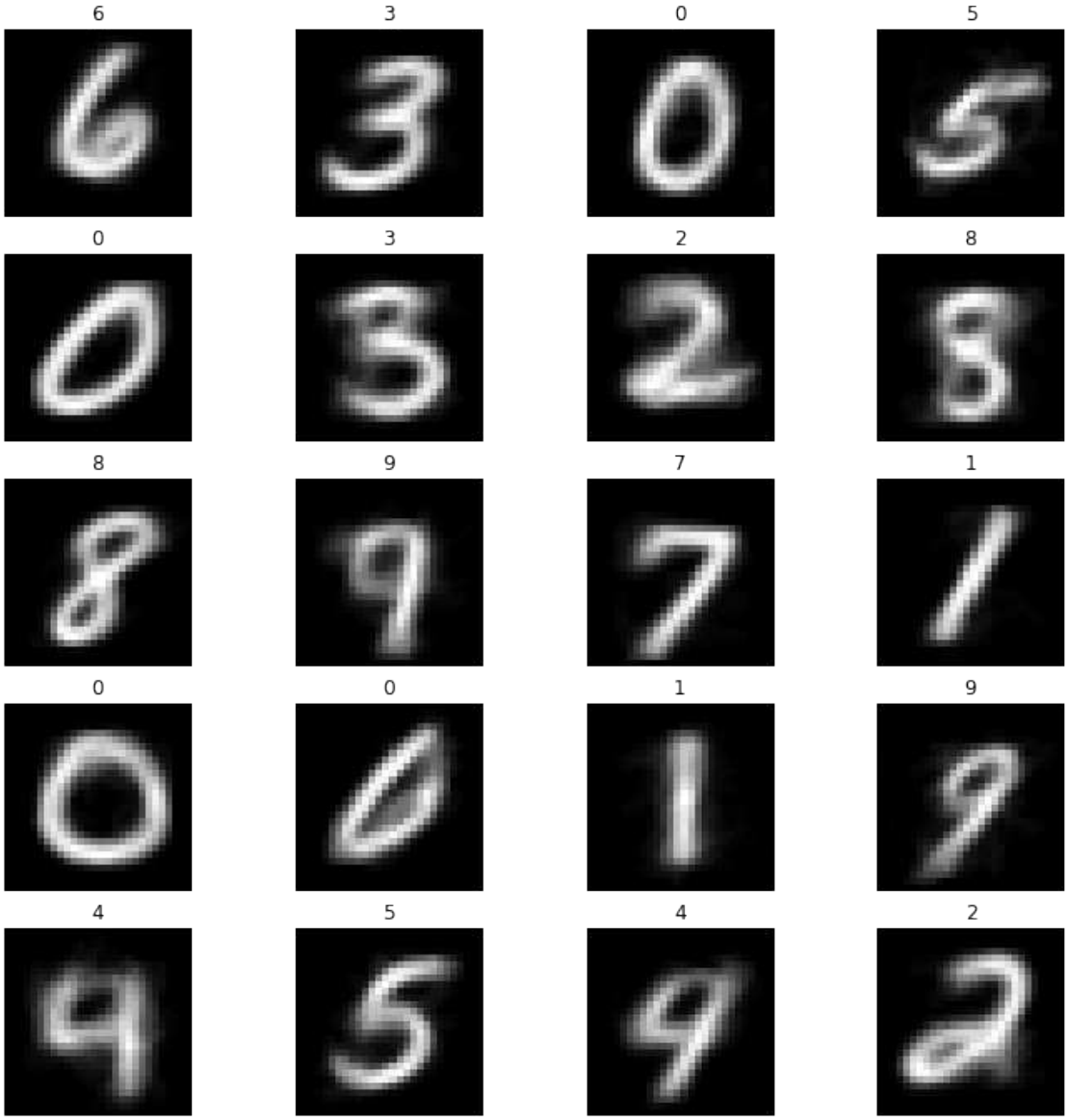
[52]:
```
{0: 6,
 1: 3,
 2: 0,
 3: 5,
 4: 0,
 5: 3,
 6: 2,
 7: 8,
 8: 8,
 9: 9,
 10: 7,
 11: 1,
 12: 0,
 13: 0,
 14: 1,
 15: 9,
 16: 4,
 17: 5,
 18: 4,
 19: 2}
```

[53]:
```
res = model.getClusterRepr().reshape((nClust,28,28))*255.0
```

[54]:
```
fig, axes = plt.subplots(5, 4, figsize = (12, 12))
plt.gray()

for i, ax in enumerate(axes.flat):
    ax.imshow(res[i])
    ax.axis('off')
    ax.set_title(refTable[i])

# display the figure
fig.show()
```



[55]:
```
ntest = 50
choices = np.random.choice(len(val_X),size=ntest)
X_test = val_X[choices]
y_test = val_y[choices]
```

[56]:
```
y_predict = np.zeros((ntest))
for i in range(ntest):
  y_predict[i] = model.predict(X_test[i])
accuracy = np.average(y_predict==y_test)
print('Accuracy = '+str(accuracy))
```

```
Accuracy = 0.7
```

[47]:
```
from tqdm import tqdm
import time
```

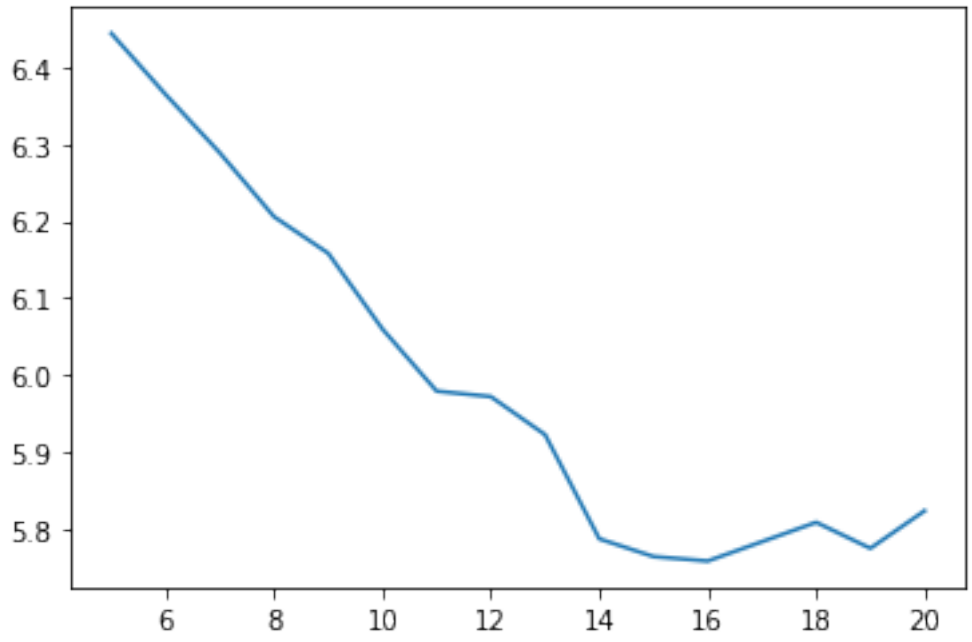
[57]:
```
#Part c
Jtable = {}
for nClus in tqdm(range(5,21)):
  model = KMeans(nClus,X,y)
  model.fitData()
  Jtable[nClus] = model.Jclus
print(Jtable)
```

```
100%|      | 16/16 [00:17<00:00,  1.12s/it]
```

```
{5: 6.4444739422753825, 6: 6.3650167594544165, 7: 6.289080436464279, 8:
6.205975545465552, 9: 6.158421597886519, 10: 6.059813407558418, 11:
5.9789508658903205, 12: 5.971842884145025, 13: 5.922322867405445, 14:
5.786795155848974, 15: 5.763709022763378, 16: 5.75795684676069, 17:
5.783116305530455, 18: 5.808294299431135, 19: 5.7744707508126885, 20:
5.82330221545483}
```

[58]:
```
plt.plot([*Jtable.keys()],[*Jtable.values()])
```

[58]:
```
[<matplotlib.lines.Line2D at 0x7f84301f8910>]
```



[49]:
```

```

[49]: