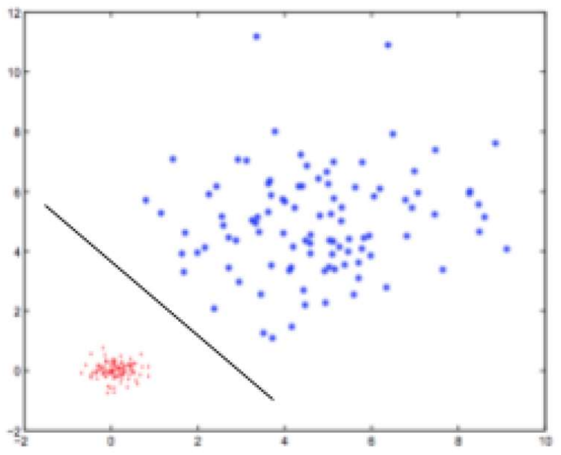
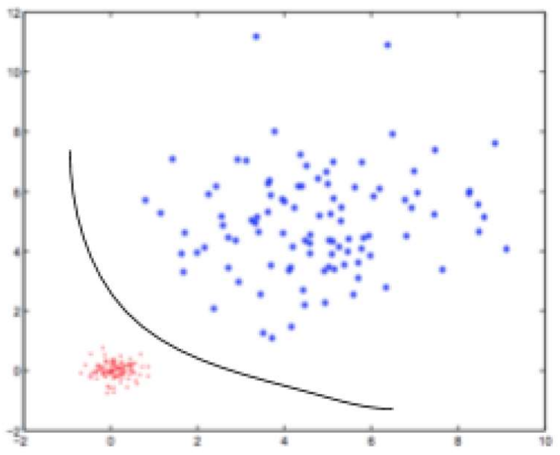
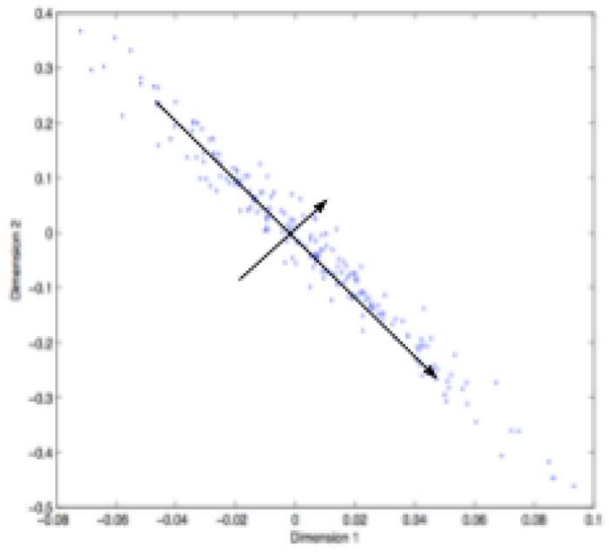
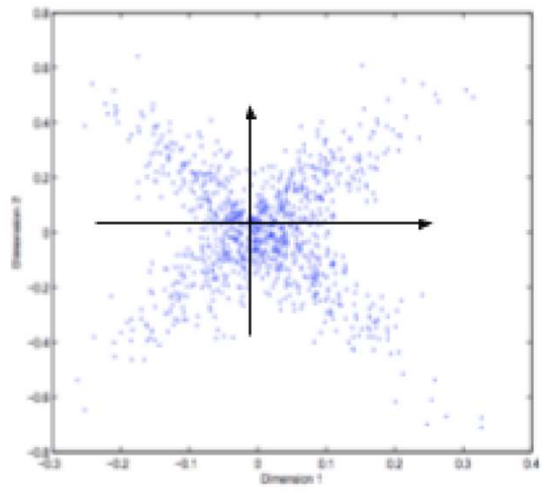
Question 1

The left one is linear, the right one is quadratic boundary.

Question 2





Question 3

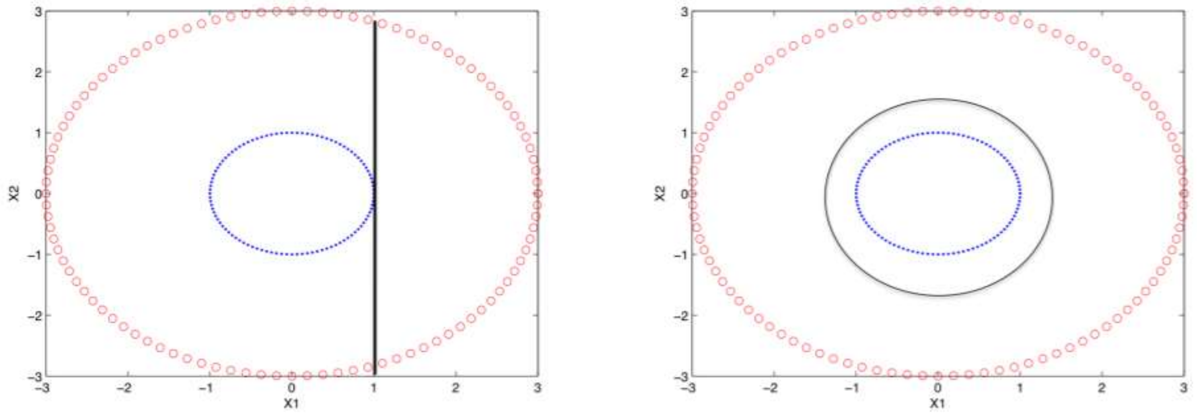
a. Ture, given data D, the probability of Hypothesis is 1.
b. False, given hypothesis H, the probability of data is not 1.
c. False, the probability of data and probability is not 1.

Question 4

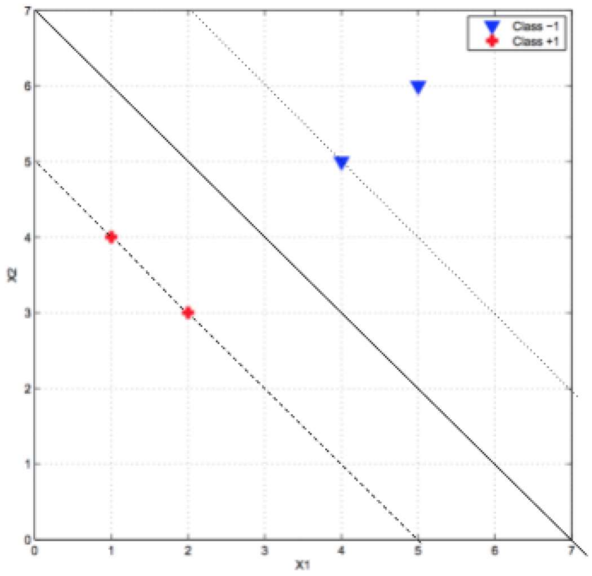
a. The maximum number of leaf nodes is 2^{k-1}, the time complexity is O(k).
b. The maximum number of leaf nodes is N, the maximum depth is N.

Question 5

a. Correct is a, the first is exp(0) = 1, the second is exp($-\infty$) = 0.
b.



c.



The boundary is y = -x+7, the SVM points are (1,4), (2,3) and (4,5).

Question 6

a. 100, it selects the variable that is significant in correlation analysis, thus the total 100 will be selected.
b. Lower, because the training data will be more.
c. Polynomial of degree 6, because it has more information that degree 4 and degree 5.

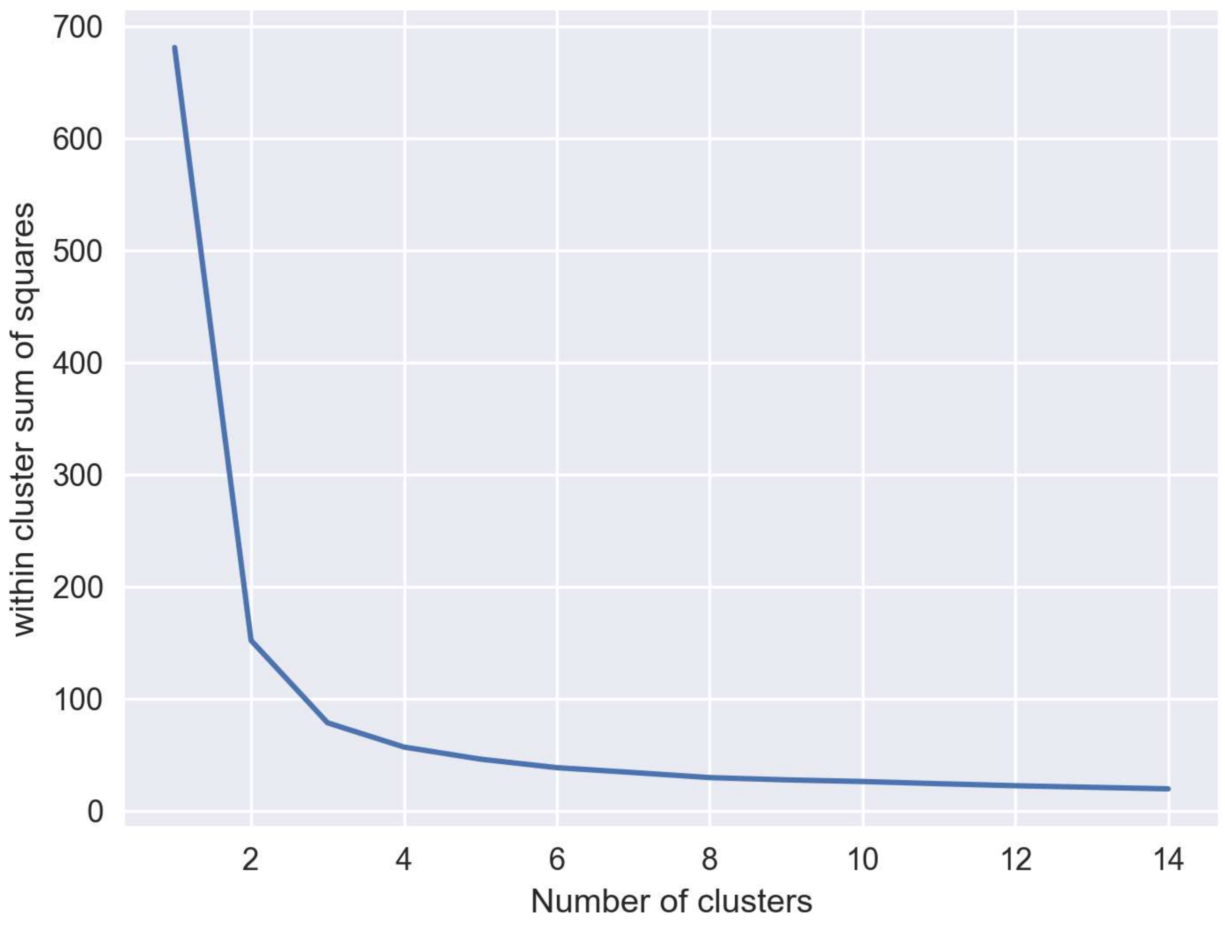
Question 7

No. It is not possible since the boundaries of decision trees are parallel to x or y axes.

Question 8

```
###Load libraries
import numpy as np
%matplotlib notebook
import matplotlib.pyplot as plt
# Though the following import is not directly being used, it is required
# for 3D projection to work
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn import datasets
import seaborn as sns; sns.set() # for plot styling
np.random.seed(5)
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

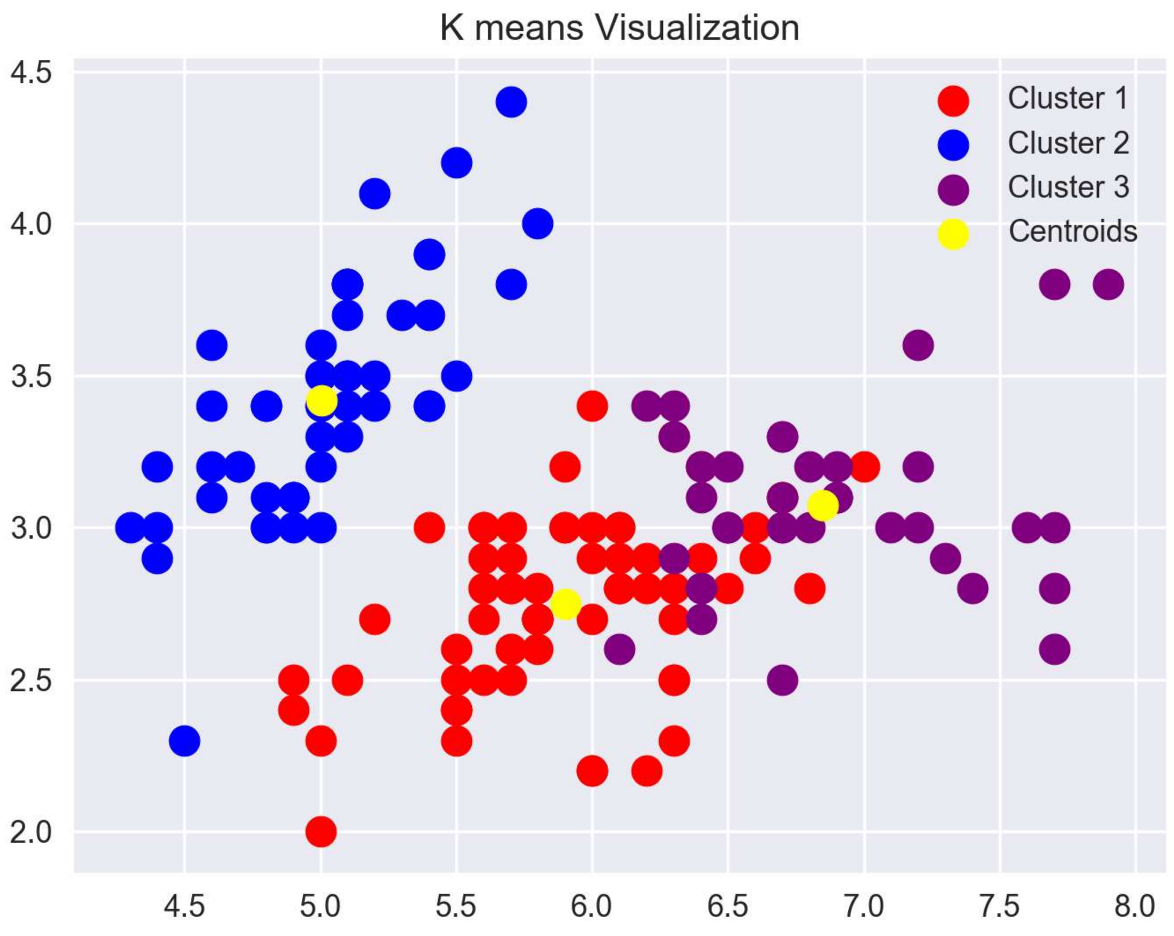
```
#Finding the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
ERR = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 500,n_init = 10, random_state = 42)
    kmeans.fit(X)
    ERR.append(kmeans.inertia_)
#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 15), ERR)
plt.xlabel('Number of clusters')
plt.ylabel('within cluster sum of squares')
plt.show()
```



From the graph above, cluster of 3 is the most optimal one.

```
#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
Y_kmeans = kmeans.fit_predict(X)
```

```
#Visualising the clusters
plt.scatter(X[Y_kmeans == 0, 0], X[Y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[Y_kmeans == 1, 0], X[Y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[Y_kmeans == 2, 0], X[Y_kmeans == 2, 1], s = 100, c = 'purple', label = 'Cluster 3')
#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1],s = 100, c = 'yellow', label = 'Centroids')
plt.title('K means Visualization')
plt.legend()
```



Question 9

```
import numpy as np
from sklearn import datasets
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

d = datasets.load_digits()
X = d.data
y = d.target
```

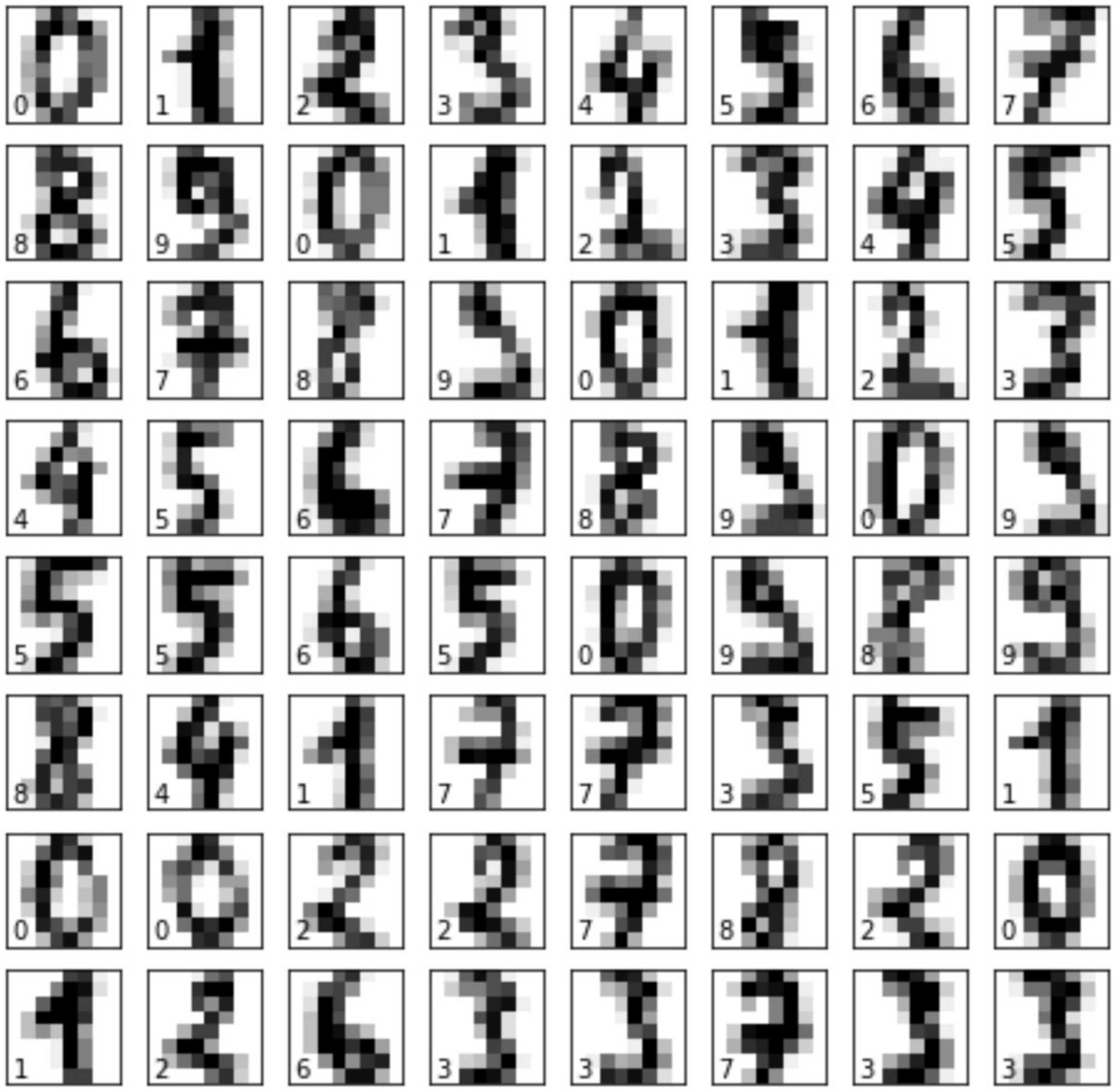
```
X.shape, y.shape
```

```
((1797, 64), (1797,))
```

```
###Split dataset into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=42)
```

```
##display the dataset
# set up the figure
fig = plt.figure(figsize=(8, 8))

for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
    ax.imshow(d.images[i], cmap=plt.cm.binary,interpolation='nearest')
    ax.text(0, 7, str(y[i]))
```



```
import pandas as pd
def accuracy(ytest, ypred):
    pred=pd.DataFrame()
    pred['label']=ytest
    pred['Predicted']=ypred
    pred['Accuracy']=1*(pred.label==pred.Predicted)
    pred = pred.groupby('label').mean()
    return pred.drop(['Predicted'], axis=1)

from sklearn.svm import SVC
ac=[]
for c in [ 0.1, 1, 10, 100]:
    for g in [5e-7, 1e-6, 2e-6]:
        clf=SVC(C=c, gamma=g)
        clf.fit(Xtrain,ytrain)
        svm_pred=clf.predict(Xtest)
        ac.append([c,g,accuracy(ytest, svm_pred).mean()])
ac
```

```
[[0.1, 5e-07, Accuracy    0.1
  dtype: float64], [0.1, 1e-06, Accuracy    0.1
  dtype: float64], [0.1, 2e-06, Accuracy    0.1
  dtype: float64], [1, 5e-07, Accuracy    0.1
  dtype: float64], [1, 1e-06, Accuracy    0.1
  dtype: float64], [1, 2e-06, Accuracy    0.1
  dtype: float64], [10, 5e-07, Accuracy    0.718732
  dtype: float64], [10, 1e-06, Accuracy    0.893875
  dtype: float64], [10, 2e-06, Accuracy    0.933628
  dtype: float64], [100, 5e-07, Accuracy    0.953352
  dtype: float64], [100, 1e-06, Accuracy    0.966069
  dtype: float64], [100, 2e-06, Accuracy    0.976435
  dtype: float64]]
```

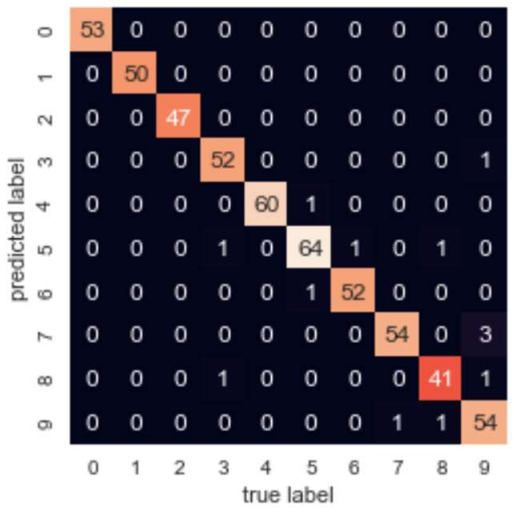
```
# Now try all samples
svm=SVC(C=100, gamma=2e-6)
svm.fit(Xtrain,ytrain)
svm_pred=svm.predict(Xtest)
```

From the above we can find that, C = 100, gamma = 2e-6 can achieve the highest accuracy.

```
svm_accuracy=accuracy(ytest, svm_pred)
print (svm_accuracy)
print (svm_accuracy.mean())
```

```
         Accuracy
label
0        1.000000
1        1.000000
2        1.000000
3        0.962963
4        1.000000
5        0.969697
6        0.981132
7        0.981818
8        0.953488
9        0.915254
Accuracy    0.976435
dtype: float64
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns; sns.set()
mat = confusion_matrix(ytest,svm_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



$$\begin{array}{c|cccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
0 & 53 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 47 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 52 & 0 & 0 & 0 & 0 & 0 & 1 \\
4 & 0 & 0 & 0 & 0 & 60 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 64 & 1 & 0 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 & 52 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 54 & 0 & 3 \\
8 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 41 & 1 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 54
\end{array}$$