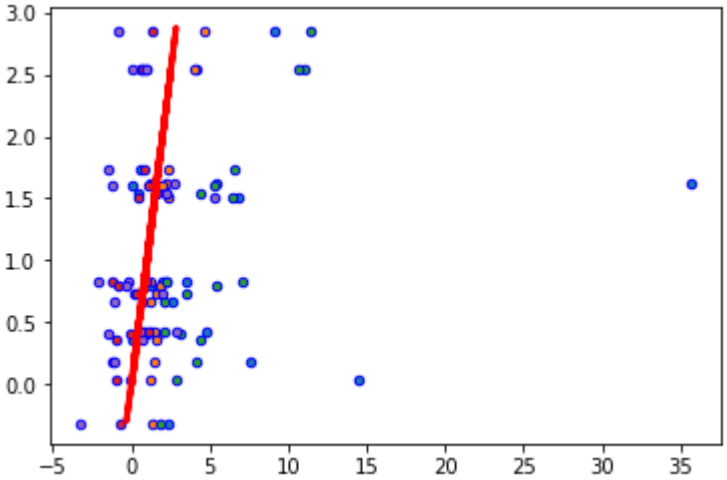
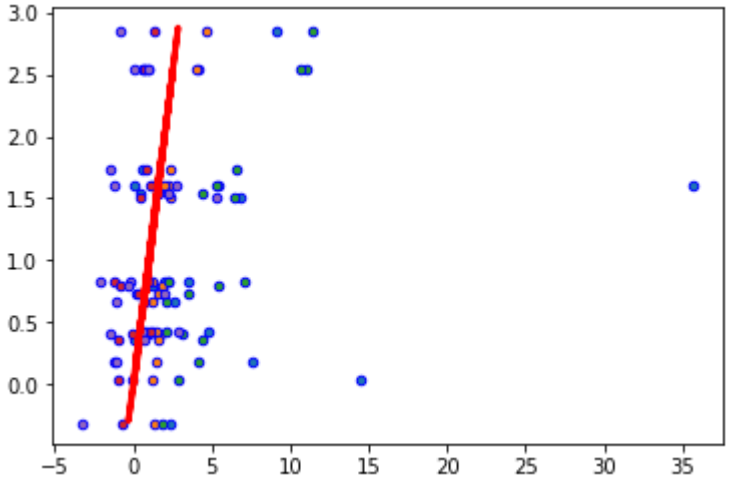
In [23]:
```
#Question 1
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import RidgeCV, LinearRegression, LassoCV
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error

#Storing data to X and y variables
data = np.loadtxt('data.csv',delimiter=',', skiprows=1)
X = data[:,:5]
y = data[:,5]

#Splitting the data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

#Creating the Linear Regression Model and fitting the training data
lin_reg  = LinearRegression().fit(X_train, y_train)

#Calculating and Printing R^2 Value for the Model
lin_r2 = lin_reg.score(X_test,y_test)
print('Linear Regression R^2 is', lin_r2)

#Get prediction for MSE and to plot graph
y_pred_lin_reg = lin_reg.predict(X_test)

#Getting MSE
print("MSE value for Linear Regression is", mean_squared_error(y_test,y_pred_lin_reg))

#Plotting Linear Regression Model Data
print("Linear Regression Model Graph")
plt.scatter(X_test[:,0], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,1], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,2], y_test ,edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,3], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,4], y_test, edgecolor='b', s=20, label='Samples')
plt.plot(y_test,y_pred_lin_reg, color='r', linewidth=3)
plt.show()

#Creating alphas variable to store range of alphas
alphas = np.logspace(-6, 6, 200)

#Ridge Regression
#Creating the Ridge Regression model and fitting the training data
rid_reg = RidgeCV(alphas = alphas, normalize = True)
rid_reg.fit(X_train, y_train)

#Calculating and Printing R^2/MSE Value for the Model
rid_r2 = rid_reg.score(X_test,y_test)
print('Ridge Regression R^2 is', rid_r2)
#Get prediction for MSE and to plot graph
y_pred_rid_reg = rid_reg.predict(X_test)
print("MSE value for Ridge Regression is", mean_squared_error(y_test,y_pred_rid_reg))

#Plotting Ridge Regression Model Data
print("Ridge Regression Model Graph")
plt.scatter(X_test[:,0], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,1], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,2], y_test ,edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,3], y_test, edgecolor='b', s=20, label='Samples')
```

```
plt.scatter(X_test[:,4], y_test, edgecolor='b', s=20, label='Samples')
plt.plot(y_test,y_pred_rid_reg, color='r', linewidth=3)
plt.show()

#Lasso Regression :
las_reg = LassoCV(alphas = alphas, normalize = True)
las_reg.fit(X_train, y_train)

#Calculating and Printing R^2/MSE Value for the Model
las_r2 = las_reg.score(X_test,y_test)
print('Lasso Regression R^2 is', las_r2)
#Get prediction for MSE and to plot graph
y_pred_las_reg = las_reg.predict(X_test)
print("MSE value for Lasso Regression is", mean_squared_error(y_test,y_pred_las_reg))

#Plotting Lasso Regression Model Data
print("Lasso Regression Model Graph")
plt.scatter(X_test[:,0], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,1], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,2], y_test ,edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,3], y_test, edgecolor='b', s=20, label='Samples')
plt.scatter(X_test[:,4], y_test, edgecolor='b', s=20, label='Samples')
plt.plot(y_test,y_pred_las_reg, color='r', linewidth=3)
plt.show()

#Code Tile to test for unseendata.csv
#Storing data to X and y variables for unseendata
unseendata = np.loadtxt('unseendata.csv',delimiter=',', skiprows=1)
unseendata_X = unseendata[:,:5]
unseendata_y = unseendata[:,5]

#Computing R^2 Value for unseendata
las_r2_unseen = rid_reg.score(unseendata_X,unseendata_y)
print("R^2 Value computed with unseendata is ", las_r2_unseen)
```

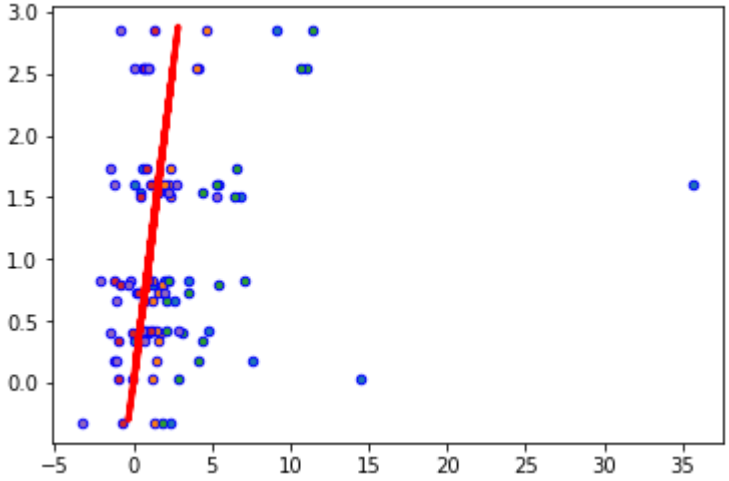
```
Linear Regression R^2 is 0.9974526591817383
MSE value for Linear Regression is 0.0019134505227723076
Linear Regression Model Graph
```




```
Ridge Regression R^2 is 0.9974576870313567
MSE value for Ridge Regression is 0.0019096738230029073
Ridge Regression Model Graph
```



```
Lasso Regression R^2 is 0.9974745868261858
MSE value for Lasso Regression is 0.0018969794395036182
Lasso Regression Model Graph
```



```
R^2 Value computed with unseendata is  0.9977381959169807
```

Justification given for your chosen regression model?

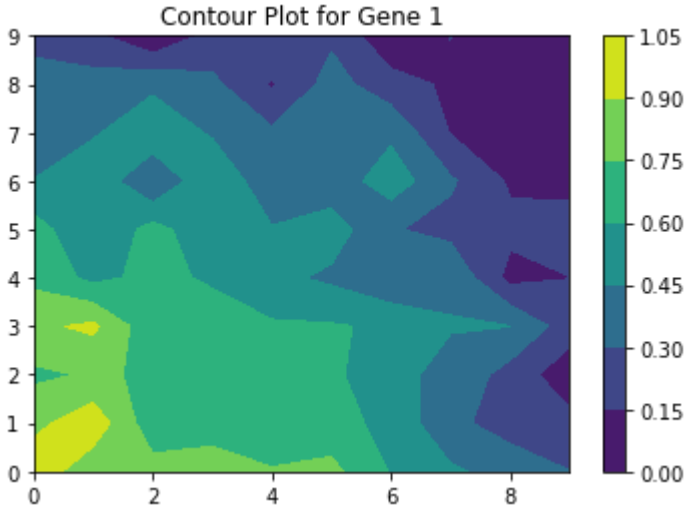
R^2 value is the coefficient of determination. It evaluates the scatter of data points around the fitted regression line. The value ranges from 0-1, 1 representing a model that explains all variation in the response variable around its mean. This means that the larger the R^2 value, the better the regression model fits the data. However, R^2 value will not tell us our model is overfitting. Therefore, Graphs were plotted to show us whether it is overfitting or underfitting.

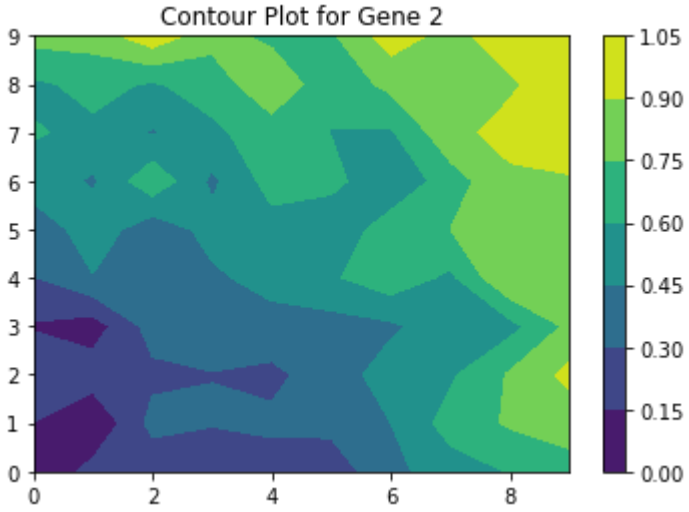
In the end,the chosen regression model would be Lasso Regression because the R^2 value computated for the test set is the most accurate compared to the Linear and Ridge Regression Models. The MSE Value for is also shown to be lower than both Linear and Ridge Regression Models. Lastly, the graph for Lasso Regression Model also shown to be not overfitting.

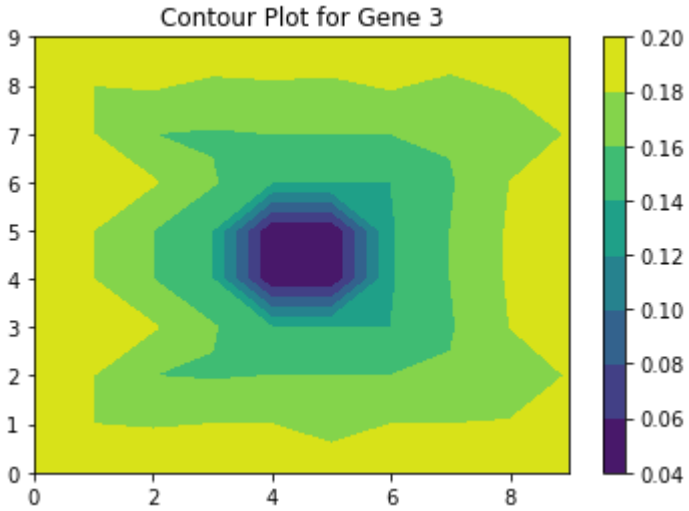
Question 2

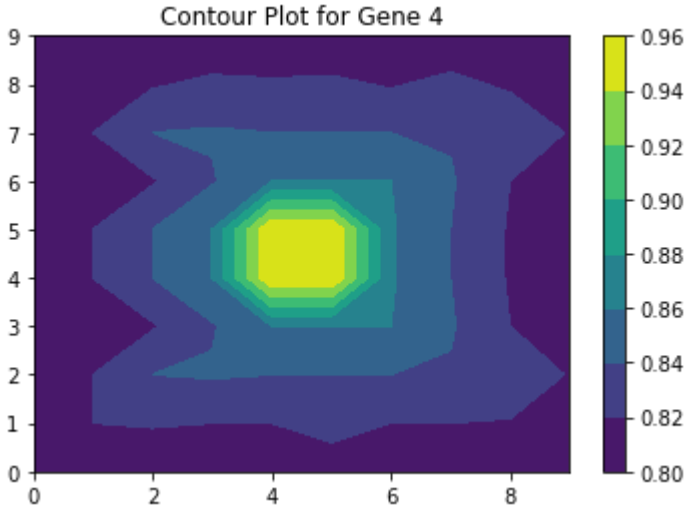
In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

#Loading the csv file and Storing data for X and Y variables
data2 = np.loadtxt('sqisland.csv',delimiter=',', skiprows=1)
```

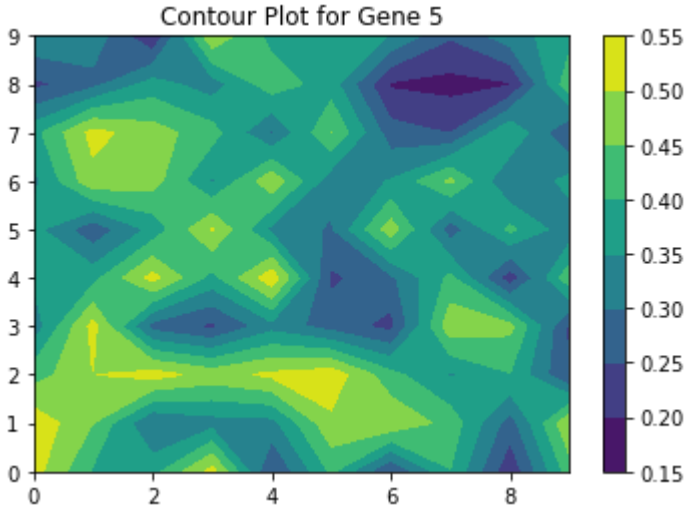
```
#Storing data for Gene1 - Gene 7 into variables and Reshaping to fit Map
G1 = data2[:,2].reshape(10,10)
G2 = data2[:,3].reshape(10,10)
G3 = data2[:,4].reshape(10,10)
G4 = data2[:,5].reshape(10,10)
G5 = data2[:,6].reshape(10,10)
G6 = data2[:,7].reshape(10,10)
G7 = data2[:,8].reshape(10,10)

#Plotting contour plot for Gene 1
cs1 = plt.contourf(G1)
plt.colorbar()
plt.title("Contour Plot for Gene 1")
plt.show()

#Plotting contour plot for Gene 2
cs2 = plt.contourf(G2)
plt.colorbar()
plt.title("Contour Plot for Gene 2")
plt.show()

#Plotting contour plot for Gene 3
cs3 = plt.contourf(G3)
plt.colorbar()
plt.title("Contour Plot for Gene 3")
plt.show()

#Plotting contour plot for Gene 4
cs4 = plt.contourf(G4)
plt.colorbar()
plt.title("Contour Plot for Gene 4")
plt.show()

#Plotting contour plot for Gene 5
cs5 = plt.contourf(G5)
plt.colorbar()
plt.title("Contour Plot for Gene 5")
plt.show()

#Plotting contour plot for Gene 6
cs6 = plt.contourf(G6)
plt.colorbar()
plt.title("Contour Plot for Gene 6")
plt.show()

#Plotting contour plot for Gene 7
cs7 = plt.contourf(G7)
plt.colorbar()
plt.title("Contour Plot for Gene 7")
plt.show()
```

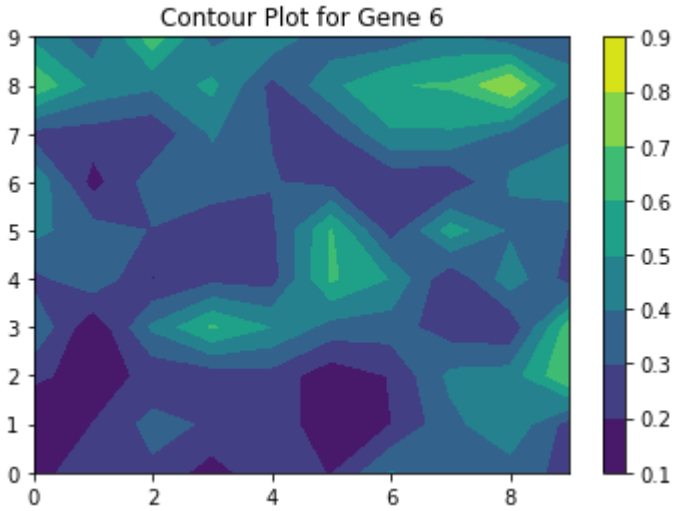

Contour Plot for Gene 1
0 1 2 3 4 5 6 7 8 9
0 2 4 6 8
1.05 0.90 0.75 0.60 0.45 0.30 0.15 0.00

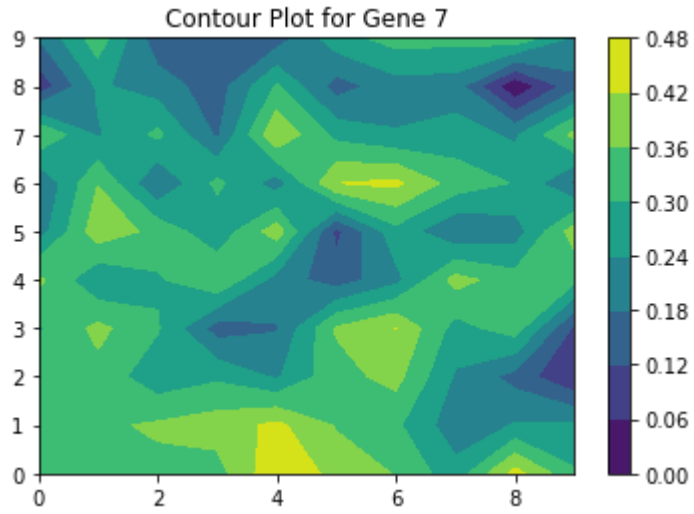

Contour Plot for Gene 2
0 1 2 3 4 5 6 7 8 9
0 2 4 6 8
1.05 0.90 0.75 0.60 0.45 0.30 0.15 0.00


Contour Plot for Gene 3
0 1 2 3 4 5 6 7 8 9
0 2 4 6 8
0.20 0.18 0.16 0.14 0.12 0.10 0.08 0.06 0.04


Contour Plot for Gene 4
0 2 4 6 8
0.96 0.94 0.92 0.90 0.88 0.86 0.84 0.82 0.80


Contour Plot for Gene 5
0 2 4 6 8
0.55 0.50 0.45 0.40 0.35 0.30 0.25 0.20 0.15


Contour Plot for Gene 6
0 2 4 6 8
0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1



1)

The Countour Plot for each Genes are plotted in sequence as titled in each plot. a) Since the hunger-gatherers are the earliest wave preceded the other two by a long margin. They are distributed uniformly across the whole island. Looking at the plots above, it shows that there are high frequency of Gene 2 in the North-Eastern parts of the island, where farmers genes would be the least considering they migrate from the South-Western corner of the island. Because of this, we can conclude that Gene 2 is the most common alleles for Locus 1 for the hunter-gatherers.

For Locus 2, since there is a way higher frequency of Gene 4 in comparison to Gene 3 as shown in the colorbar of the plots above (Range of 0.8-0.96 vs Range of 0.04-0.2), we can concluded that Hunter-Gatherers most common alleles for Locus 2 is Gene 4.

b) Looking at the plots above, the frequency of Gene 2 is most common in the North-Eastern parts of the island whereas Gene 1 is most common in the South-Western parts of the island. This shows Gene 1 is the most common alleles for Locus 1 for the farmers since it was where they Migrated to the island.

For Locus 2, this would be similar to the case of the hunter-gatherers since that there are a large frequency of Gene 4 in comparison to Gene 3. Eventhough towards the South-Western corner of the island, there is no significant increase in Gene 3. Hence, we can also conclude that the most common alleles for Locus 2 for the farmers is Gene 4

2) For Locus 1, looking at the plots above, it shows that the seafarers genes are evenly distributed since there are no patterns that can be seen along the coast(which are where the seafarers are).

For Locus 2, it shows that the relative frequency for Gene 3 is increasing around the coast, where the seafarers are, this means that Gene 3 is the more common alleles for Locus 2.

For Locus 3, it seems like there are not much pattern to be seen around the coast from the plots above. Therefore, we can not conclude anything.

In [3]:
```
#Storing variables counter and genes to be used in for loop
genes = [G1,G2,G3,G4,G5,G6,G7]
counter = 0
```

```
#total = 0

#for loop to calculate and display variance for each Gene
for i in genes:
    counter += 1
    variance = np.var(i)
    #total += variance
    print("Variance of Gene", counter, "is", variance)

#print(total)
```

```
Variance of Gene 1 is 0.060582434371
Variance of Gene 2 is 0.06058243437099999
Variance of Gene 3 is 0.0010836736
Variance of Gene 4 is 0.0010836735999999985
Variance of Gene 5 is 0.009514104445105358
Variance of Gene 6 is 0.020771410865005658
Variance of Gene 7 is 0.009074831617783594
```

In [4]:
```
#Calculating Pearson correlation for Gene 1 and 4
from scipy import stats
G1_cor = data2[:,2]
G4_cor = data2[:,5]
pearson_cor1 = stats.pearsonr(G1_cor, G4_cor)
print("Pearson correlation between Gene 1 and 4 is ",pearson_cor1[0])

G5_cor = data2[:,6]
pearson_cor2 = stats.pearsonr(G1_cor, G5_cor)
print("Pearson correlation between Gene 1 and 5 is ",pearson_cor2[0])
```

```
Pearson correlation between Gene 1 and 4 is  0.07526456660171582
Pearson correlation between Gene 1 and 5 is  0.3601203240798583
```

The null hypothesis of non-correlation can not be rejected for either at the significant level because the Pearson correlation have to be < 0.05 which in this case either pair did not satisfy it.

No, this results will not agree with the previous hypothesis since the null hypothesis can not be rejected. This means that there exist some correlation between (Gene 1 and 4) and (Gene 1 and 5).

In [15]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale, normalize

#Getting the gene data
G1 = data2[:,np.newaxis,2]
G2 = data2[:,np.newaxis,3]
G3 = data2[:,np.newaxis,4]
G4 = data2[:,np.newaxis,5]
G5 = data2[:,np.newaxis,6]
G6 = data2[:,np.newaxis,7]
G7 = data2[:,np.newaxis,8]

#Concatenate all data into one
G_all = np.concatenate((G1,G2,G3,G4,G5,G6,G7), axis=1)

#Normalizing the data
Gn = normalize(G_all)

#Applying PCA
pca = PCA(n_components=7)
G_r = pca.fit_transform(Gn)
```

```
#Storing variables counter and total to be used in for loop
total = 0
counter = 0

#Calculating and Displaying variance for new transformed data
for i in range(7):
    counter += 1
    variance = np.var(G_r[:,i])
    total += variance
    print("New Variance of Gene", counter, "is", variance)

#Printing total sum of variance
print("Total sum of Variance ",total)
```

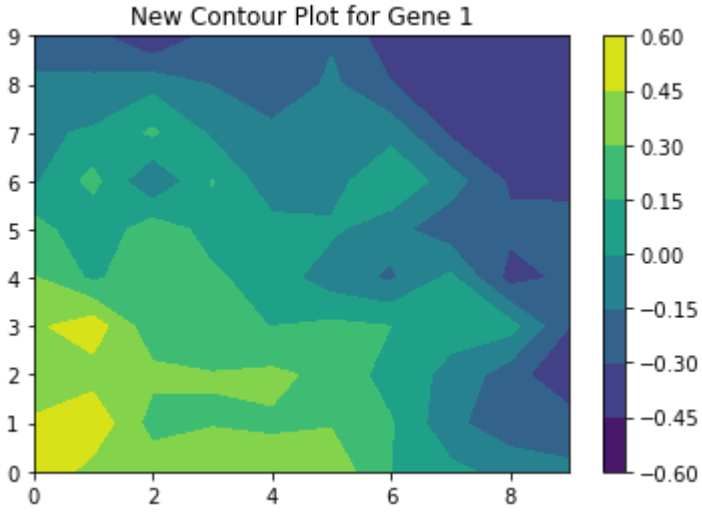
```
New Variance of Gene 1 is 0.0707586878681787
New Variance of Gene 2 is 0.012113149474097213
New Variance of Gene 3 is 0.0047898081237278765
New Variance of Gene 4 is 0.001677258012579065
New Variance of Gene 5 is 0.0007792660257549584
New Variance of Gene 6 is 2.8753617303469614e-20
New Variance of Gene 7 is 9.517653620041751e-34
Total sum of Variance  0.0901181695043378
```

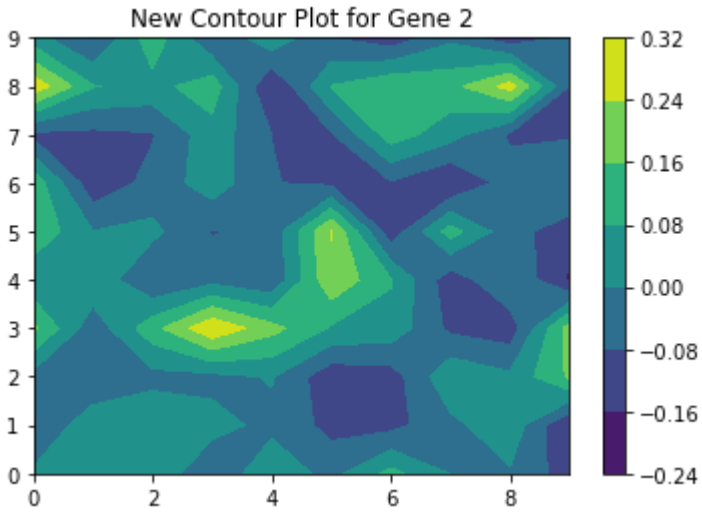
Old Sum of Variance = 0.16269256286989464 New Sum of Variance = 0.0901181695043378

Sum of variance is essentially the average squared value from the mean(expected value). Since the new sum of variance is only 0.09, it indicates that the points tend to be close to the mean. This means that the result can be expected.

In [21]:

```
#Reshaping data so that it fits the map
G_r1 = G_r[:,0].reshape(10,10)
G_r2 = G_r[:,1].reshape(10,10)

#Plot contour plot for first two PCA components
new_cs1 = plt.contourf(G_r1)
plt.colorbar()
plt.title("New Contour Plot for Gene 1")
plt.show()

new_cs2 = plt.contourf(G_r2)
plt.colorbar()
plt.title("New Contour Plot for Gene 2")
plt.show()
```





From the result shown in the first two PCA components, the two pairs of plots before the PCA would be more helpful for the task. This is because the data transformed are using all 7 principal components, so there are no reduced in dimensionality. The PCA tries to find new variables amongst the data. However, since we already have information for relative frequency of all 7 of the Genes, PCA would not help much for interpreting the data if using all 7 principal components.

For a realistic data set with hundreds of genes, assuming we still keep our principal components to 7, we would find the data after PCA to be more helpful since PCA's main purpose is to reduce the dimensionality of a dataset, while preserving as much information as possible. Having 7 principal components instead of having to analyse of hundreds of genes would help us interprete the data better and easier.