```
157      values, vectors = la.eig(A, B)
158      vectors = nu.array([ vector.reshape(2,1) / nu.linalg.norm(vector) for vector in ve
159
160      return Eigen(values, vectors)
161
162
163  class Projection(ClassData):
164      def __init__(self, classData, vector):
165          vector = vector.reshape(2,1)
166          ndarrayPoints = nu.array([ nu.dot(row.reshape(1,2), vector) * vector for row i
167          self.points = [ XYPoint([float(ndarrayPoint[0]), float(ndarrayPoint[1])]) for
168          self.len = classData.len
169
170
171  def pricipalComponentOf(class1, class2):
172      A = covarianceTotal(class1, class2)
173      B = nu.eye(2)
174      values, vectors = la.eig(A, B)
175      vectors = nu.array([ vector.reshape(2,1) / nu.linalg.norm(vector) for vector in ve
176      return Eigen(values, vectors)
177
178
179
180  # NOTE: - layout of results:  ************************************************  layo
181
182  def errataOf(checkList, correctList):
183      checkArray = nu.array(checkList)
184      correctArray = nu.array(correctList)
185      return checkArray == correctArray
186
187
188  def recognitionRateOf(errata):
189      intUniversalFunc = nu.frompyfunc(int, 1, 1)
190      return intUniversalFunc(errata).sum() / len(errata)
191
192
193
194  # NOTE: recognition lines:  ************************************************  reco
195
196  RecogLine = namedtuple('RecogLine', 'polyExpr vectorExpr')
197
198  def recogLineOfEu(class1, class2):
199      vector = nu.flipud(class1.mean - class2.mean)
200      vector[0,0] = -vector[0,0]
201
202      x, y = sy.symbols('x,y')
203      midlePoint = (class1.mean + class2.mean) / 2.0
204      poly = y - midlePoint[1,0] - vector[1,0]/vector[0,0] * (x - midlePoint[0,0])
```