P4 - Códigos

Função para geração de distribuição de pontos em formato circular uniforme.

```
def createCircularDistribution(x, y, r, n):
    ### x, y and r must have the same length.
    ### x and y are vectors with the center coordinates of each circle
    ### and r is a vector with the radius of each circle.
    ### n is the number of points
    length = n*len(r)
    data = np.zeros((length , 2))
    counter = 0
    for i in range(len(r)):
        for _ in range(n):
            randX = np.random.uniform(-r[i], r[i]) + x[i]
            randY = np.random.uniform(-r[i], r[i]) + y[i]
            while((((randX-x[i])**2 + (randY-y[i])**2)**(0.5) > r[i])):
                randX = np.random.uniform(-r[i], r[i]) + x[i]
                randY = np.random.uniform(-r[i], r[i]) + y[i]
            data[counter][0] = randX
            data[counter][1] = randY
            counter += 1

    return(data)
```

Tal função recebe os vetores x, y, r e n que são respectivamente, as coordenadas das distribuições de pontos x e y, o raio de cada uma das distribuições e o número de pontos final.

Funções para transformação do pontos.

```
def displace3(x, y):

    return [x+(-x**2 * np.sign(x)/2), y+(-y**2 * np.sign(y)/2)]

#def displace4(x, y):

 #   s = 0.3**2
 #   c = [[0.25, 0.25], [-0.25, -0.25]]
    # for i in range(len(c)):
    #     x = np.exp(-( (x-c[i][0])**2) /(s) + y-c[i][1])**2) /(s))
    #     y = np.exp(-( (x-c[i][0])**2) /(s) + y-c[i][1])**2) /(s))

 #   return [x, y]

def displace5(x, y):

    return [np.power(-np.sign(x)+x, 3) ,0]

def displace6(x, y):

    xMax = x.max()
    xMin = x.min()

    yMax = y.max()
    yMin = y.min()

    return [(x-xMin)/(xMax-xMin), (y-yMin)/(yMax-yMin)]

def displace8(x, y):

    xStd = np.std(x)
    yStd = np.std(y)

    xMean = np.mean(x)
    yMean = np.mean(y)

    return [(x-xMean)/(xStd), (y-yMean)/(yStd)]
```

Em cada uma delas, implementamos as transformações em x e em y demonstradas no relatório de resultados.

Função para geração das imagens contendo os pontos origais, o campo vetorial e os pontos transformados.

```
def displacementModification(x, y, r):

    ###Generating circles
    circles = createCircularDistribution(x, y, r,500)

    x = circles[:, 0]
    y = circles[:, 1]

    ###Points after transformation by Displacement Filed
    data = displace3(x, y)

    xbar = data[0]
    ybar = data[1]

    ####Displacement Field

    grid = [-1, 1, 0.1]

    X, Y = np.meshgrid(np.arange(grid[0], grid[1], grid[2]), np.arange(grid[0], grid[1], grid[2]))

    fieldData = displace3(X,Y)

    u = fieldData[0]
    v = fieldData[1]

    my_dpi=96

    f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=True, figsize=(2400/my_dpi, 600/my_dpi), dpi=my_dpi)
    # my_dpi=96
    # plt.figure(figsize=(1000/my_dpi, 1000/my_dpi), dpi=my_dpi)

    # ax1.set_aspect('equal', adjustable='box')
    # ax2.set_aspect('equal', adjustable='box')
    # ax3.set_aspect('equal', adjustable='box')

    box = [grid[0]+grid[0]*0.1, grid[1]+grid[1]*0.1]

    ax1.set_xlim(box)
    ax1.set_ylim(box)
    ax1.set_xlabel("x")
    ax1.set_ylabel("y")
    ax1.set_title("(a)")


    ax2.set_xlim(box)
    ax2.set_ylim(box)
    ax2.set_xlabel("x")
    ax2.set_ylabel("y")
    ax2.set_title("(b)")


    ax3.set_xlim(box)
    ax3.set_ylim(box)
    ax3.set_xlabel("x")
    ax3.set_ylabel("y")
    ax3.set_title("(c)")

    ax1.scatter(x, y, color='orange', label = "Circle", s=3, )
    ax2.quiver(X, Y, u, v, units='width', color="blue", label="Displacement Field")
    ax3.scatter(xbar, ybar, color='gray', label = "Modified Circle", s=3)

    #plt.quiver(X, Y, u, v, units='width', color="blue", label="Displacement Field")
    #plt.scatter(x, y, color='orange', label = "Circle", s=3)
    #plt.scatter(x+xbar, y+ybar, color='gray', label = "Modified Circle", s=3)
```

```
    #plt.legend()
    #plt.savefig('displace6.png', bbox_inches='tight')
    plt.show()
```

Função para a realização do método do PCA.

```
def pca(x, y, r):

    #create circle of points:
    circles = createCircularDistribution(x, y, r, 2000)

    x = circles[:, 0]
    y = circles[:, 1]

    ##alongate
    y2 = 0.4*y

    ##Rotate
    mat = np.dot([[np.cos(np.pi/6), np.sin(np.pi/6)], [np.sin(np.pi/6), np.cos(np.pi/6)]], [x, y2])

    k = np.cov(mat)
    eValue, eVector = np.linalg.eig(k)

    normalSort = np.argsort(eValue)

    print(eValue)
    print(eVector)
    ###Decreasing sorted
    eValue = np.flip(eValue[normalSort])
    eVector = np.flip(eVector[normalSort], axis=0)


    print(eValue)
    print(eVector)

    my_dpi=96
    fig, ax = plt.subplots(1, 1, figsize=(2400/my_dpi, 600/my_dpi), dpi=my_dpi)
    # fig.suptitle("Principal Component Analysis")
    ax.set_title("$\eta = $" + str(np.round(eValue[0]/(eValue[0]+eValue[1]), 4))+ "%")
    offset = 3
    ax.axis('equal')
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.scatter(x,y, color = "darkorange", s = 3, label="Original data")
    ax.scatter(x+offset,y2, color = "dimgray", s = 3, label="Alongated data")
    ax.scatter(mat[0]+2*offset, mat[1], color = "dimgray", s = 3)
    ax.quiver(*[2*offset,0], *eVector[:,0], width = 0.004, color = "orange", label= "$\lambda_1 = %.3f$"%eValue[0])
    ax.quiver(*[2*offset,0], *eVector[:,1], width = 0.004, color = "black", label= "$\lambda_2 = %.3f$"%eValue[1])
    plt.legend()
    plt.show()
```

Novamente, a função *pca* toma como atributos os pontos x, y e r, cria tal distribuição, modifica-a utilizando uma matriz de rotação e achatamento e segue calculando o PCA conforme as operações de craiação da matriz de covariância, calculo dos autovetores e autovalores, e organização em ordem decrescente.