Transformación isométrica afín

Sergio González Montero (U2)

6 de mayo de 2024

1. Objetivo

Dado un sistema con N elementos, $S = \{a^j, (x^j, y^j, \ldots)\}_{j=1}^{N}$, se considera la transformación isométrica afín $R_\theta^{(xy)}$ aplicada en torno al centroide del sistema y la traslación $v = (v_1, v_2, \ldots)$, todo ello con la métrica euclídea. Se aplicará dicha transformación a una figura tridimensional: rotación de $\theta = 3\pi$ y traslación $v = (0, 0, d)$, siendo d el diámetro del sistema S. Además, del sistema representado por la imagen *hurricane-isabel.png*, se considera el subsistema σ dado por el tercer color de RGB, azul, entre 100 y 255. Se localizará el centroide y se aplicará la transformación anterior pero con $\theta = 6\pi$ y $v = (d, d, 0)$.

2. Material y datos

Se usaron los siguientes apartados de las notas de Robert Monjo de la asignatura de Geometría Computacional, a día 26/04/2024: Sección 4.1.3. Espacio afín y difeomorfismos afines; Definición 4.1.6. Baricentro; Definición 4.1.7. Transformación afín, Definición 4.1.8. Transformación isométrica afín, Sección 4.1.4. Rotaciones, reflexiones y traslaciones y Algoritmo 4.1.1. Transformación isométrica de imágenes.

En cuanto al código, se usó como base la plantilla *GCOM2024-practica4_plantilla*. En él, se han utilizado las librerías siguientes:

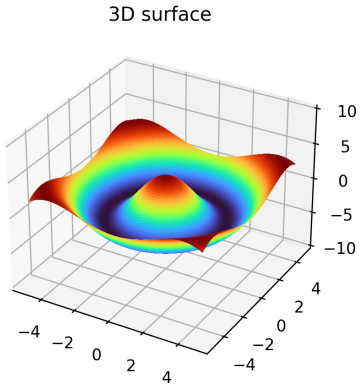
1. numpy: operaciones matemáticas y cálculos numéricos
2. matplotlib: para visualización de gráficos y *GIF*
3. scypy.spatial: añade el cálculo y representación del diagrama de la envolvente convexa
4. OS: para manipular rutas de archivos y directorios
5. skimage: procesamiento de imágenes en Python

3. Resultados

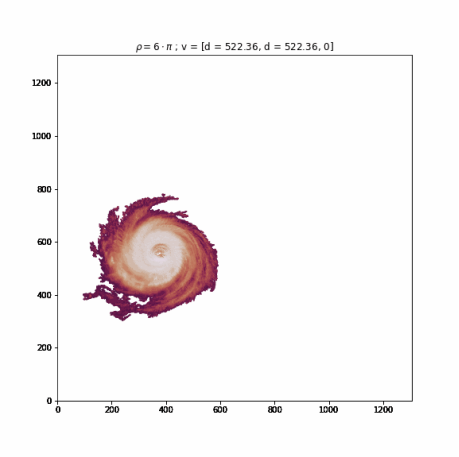
El baricentro del sistema del primer apartado se sitúa en el lugar con coordenadas $(X, Y, Z) = (-0{,}05, -0{,}05, -0{,}588)$ en torno al cual se aplica la transformación isométrica afín, con un desplazamiento en el eje Z igual al diámetro del propio sistema, siendo este $d = 14{,}0$. Para el sistema correspondiente a *hurricane-isabel.png*, el baricentro se sitúa en las coordenadas $(X, Y) = (291{,}184, 466{,}961)$, con un desplazamiento diagonal de longitud igual al diámetro del sistema, $d = 522{,}361$.

Los baricentros han sido calculados con la media de cada factor del sistema y ambos diámetros han sido calculados mediante envolventes convexas.

Se incluirán los *GIF* correspondientes en la subida al Campus Virtual.



(a) Figura tridimensional, el sistema



(b) *hurricane-isabel.png* a tiempo t_i

Figura 1: 3D figure y *hurricane-isabel.png* en un tiempo t_i

4. Conclusión

Se observa que el cálculo del diámetro de un sistema como los planteados en la práctica es mucho más eficiente en tiempo si se hace mediante la envolvente convexa que si se hace por fuerza bruta. Cabe destacar la sutileza de aplicar la transformación en torno al centroide, pues hay que restar dichas coordenadas al introducirlo en la función de animación pero sumárselas de nuevo en la función de transformación para hacerlo como se desea. De otra manera (salvo operaciones análogas), no resultaría el desplazamiento pedido.

5. Código

Programa 1: practica4.py

```
# -*- coding: utf-8 -*-

# Sergio Gonzalez Montero
# Victor Martin Martin

import numpy as np
from numpy import cos as cos, sin as sin, pi as pi, sqrt as sqrt
import matplotlib.pyplot as plt
from matplotlib import animation
from matplotlib import cm
import os

from scipy.spatial import ConvexHull
from scipy.spatial.distance import cdist
from skimage import io

print("-----------------3D SURFACE-----------------")
fig = plt.figure(figsize=plt.figaspect(1))

ax = fig.add_subplot(1, 1, 1, projection='3d')

X = np.arange(-5, 5, 0.1)
Y = np.arange(-5, 5, 0.1)
X, Y = np.meshgrid(X, Y)
R = -sqrt(X**2 + Y**2)
Z = 3*cos(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.turbo,
                       linewidth=0, antialiased=False)
ax.set_zlim(-10, 10)
plt.title("3D surface")
plt.savefig('3D surface.png', dpi=250)

plt.show()

print("\n----------------ROTATION AND TRANSLATION----------------\n")

def bar3D(x,y,z):
    """
    x, y, z : coordenadas de los puntos del sistema
    Se calcula el baricentro del sistema
    """
    print("Calculating the barycenter...")
```

```
43      barx, bary, barz = np.mean(x), np.mean(y), np.mean(z)
44
45      return (barx, bary, barz)
46
47  def diametro3D(x, y, z):
48      """
49      x, y, z : coordenadas de los puntos del sistema
50      Se calcula el diametro del sistema
51      """
52      print("\nCalculating the diameter...")
53      xaux = x.ravel()
54      yaux = y.ravel()
55      zaux = z.ravel()
56      # Calcula todas las distancias al cuadrado
57      distances_squared = (xaux[:, None] - xaux)**2 + \
58                          (yaux[:, None] - yaux)**2 + \
59                          (zaux[:, None] - zaux)**2
60
61      # Ignora las distancias entre el mismo punto
62      np.fill_diagonal(distances_squared, 0)
63      # Encuentra la distancia maxima al cuadrado
64      max_distance_squared = distances_squared.max()
65      d = np.sqrt(max_distance_squared)
66
67      return d
68
69  def trans_iso_afin3D(x, y, z, M, v):
70      '''
71      x, y, z : coordenadas de los puntos del sistema
72      M : matriz de rotacion
73      v : vector de traslacion
74      '''
75      lenx = len(x)
76      xt = np.zeros(shape=(lenx,lenx))
77      yt = np.zeros(shape=(lenx,lenx))
78      zt = np.zeros(shape=(lenx,lenx))
79
80      for i in range(len(x)):
81          for j in range(len(x)):
82              q = np.array([x[i][j], y[i][j], z[i][j]])
83              xt[i][j], yt[i][j], zt[i][j] = np.matmul(M,q) + v
84
85      return xt, yt, zt
86
87  barx, bary, barz = bar3D(X,Y,Z)
88  print("Barycenter:", round(barx, 3),",", round(bary, 3),",", round(barz, 3))
```

```
d = diametro3D(X,Y,Z)
print("Diameter:", round(d, 2))

def animate3D(t):
    """
    Creacion del GIF con paso de frames t
    """
    theta = 3*pi*t
    ro = np.array([[cos(theta), -sin(theta), 0],
                   [sin(theta), cos(theta),  0],
                   [0,          0,           1]])

    v = np.array([0, 0, d])*t
    print(f"\nt = {round(t,3)}\n ro = \n{ro}\n v = {v}\n")
    ax = plt.axes(xlim=(-8,8), ylim=(-8,8), zlim=(barz-2,d+2),
                  projection='3d')

    x, y, z = trans_iso_afin3D(X-barx, Y-bary, Z-barz, ro, v)
    ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.turbo,
                            linewidth=0, antialiased=False)
    return ax,

def init3D():
    return animate3D(0),


fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
print("Making the GIF...")
ani = animation.FuncAnimation(fig, animate3D, frames=np.arange(0,1,0.025),
                              init_func=init3D, interval=20)
plt.title(rf"$\rho = 3*\pi$ ; v = [0, 0, d = {round(d, 2)}]")
ani.save("affine isometric transformation.gif", fps = 10)
print("Done\n")


print("----------------HURRICANE ISABEL TRANSFORMATION----------------\n")

def bar2D(x,y):
    """
    x, y : coordenadas de los puntos del sistema
    Se calcula el baricentro del sistema
    """
    print("Calculating the barycenter...")
    barx, bary = np.mean(x), np.mean(y)

```

```
135     return (barx, bary)
136
137 def diametro2D(x, y):
138     """
139     x, y, z : coordenadas de los puntos del sistema
140     Se calcula el diametro del sistema
141     """
142     print("\nCalculating the diameter...")
143     # Envolvente convexa
144     points = np.array([x,y]).transpose()
145     hull = ConvexHull(points)
146     # Extraccion de los puntos de la envolvente
147     hullpoints = points[hull.vertices,:]
148
149     # La mayor distancia entre los puntos de la envolvente
150     hdist = cdist(hullpoints, hullpoints, metric='euclidean')
151     bestpair = np.unravel_index(hdist.argmax(), hdist.shape)
152
153     p1 = hullpoints[bestpair[0]]
154     p2 = hullpoints[bestpair[1]]
155
156     d = np.sqrt((p1[0]-p2[0])**2+(p1[1]-p2[1])**2)
157
158     return d
159
160 ruta = "D:/2023-2024/SEGUNDO CUATRI/GCOM/Practica_4"
161 os.getcwd()
162 os.chdir(ruta)
163
164 img = io.imread("hurricane-isabel.png")
165
166 fig = plt.figure(figsize=(5,5))
167 p = plt.contourf(img[:,:,2], cmap=cm.twilight,
168                  levels=np.arange(100,255,2))
169 plt.axis('off')
170
171 xyz = img.shape
172
173 x = np.arange(0,xyz[0],1)
174 y = np.arange(0,xyz[1],1)
175 xx,yy = np.meshgrid(x, y)
176 xx = np.asarray(xx).reshape(-1)
177 yy = np.asarray(yy).reshape(-1)
178 z  = img[:,:,2]
179 z = np.transpose(z)
180 zz = np.asarray(z).reshape(-1)
```

```
181
182 # Variables de estado coordenadas, azul >= 100
183 x0 = xx[zz>100]
184 y0 = yy[zz>100]
185 z0 = zz[zz>100]/zz.max()
186 # Variable de estado: color
187 col = plt.get_cmap("twilight")(np.array(z0))
188
189 barx1, bary1 = bar2D(x0, y0)
190 print("Barycenter:", round(barx1, 3),",", round(bary1, 3))
191 d1 = diametro2D(x0, y0)
192 print("Diameter:", round(d1, 3))
193
194 def transf2D(x, y, z, M, v):
195     '''
196     x, y, z : coordenadas de los puntos del sistema
197     M : matriz de rotacion
198     v : vector de traslacion
199     '''
200     lenx = len(x)
201     xt = np.zeros(lenx)
202     yt = np.zeros(lenx)
203     zt = np.zeros(lenx)
204
205     for i in range(len(x)):
206         q = np.array([x[i], y[i], z[i]])
207         xt[i], yt[i], zt[i] = np.matmul(M, q) + v + (barx1, bary1, 0)
208
209     return xt, yt, zt
210
211 def animate2D(t):
212     """
213     Creacion del GIF con paso de frames t
214     """
215     theta = 6*pi*t
216     ro = np.array([[cos(theta), -sin(theta), 0],
217                    [sin(theta), cos(theta),  0],
218                    [0,          0,           1]])
219
220     v = np.array([d1, d1, 0]) * t
221     print(f"\nt = {round(t, 3)}\n ro = \n{ro}\n v = {v}\n")
222
223     ax = plt.axes(xlim=(0, 2.5*d1), ylim=(0, 2.5*d1))
224
225     XYZ = transf2D(x0-barx1, y0-bary1, z0, ro, v)
226
```

```
227     col = plt.get_cmap("twilight")(np.array(XYZ[2]))
228     ax.scatter(XYZ[0], XYZ[1], c=col, s=0.1, animated=True)
229 
230     return ax,
231 
232 def init2D():
233     return animate2D(0),
234 
235 fig = plt.figure(figsize=(8,8))
236 ax = fig.add_subplot(111, projection='3d')
237 print("\nMaking the GIF...")
238 ani = animation.FuncAnimation(fig, animate2D, frames=np.arange(0,1,0.025),
239                               init_func=init2D, interval=20)
240 plt.title(rf"$\rho = 6\cdot\pi$ ; v = [d = {round(d1, 2)}, d = {round(d1, 2)}, 0]")
241 os.chdir(ruta)
242 ani.save("hurricane isabel transformation.gif", fps = 10)
243 print("Done")
244 os.getcwd()
```