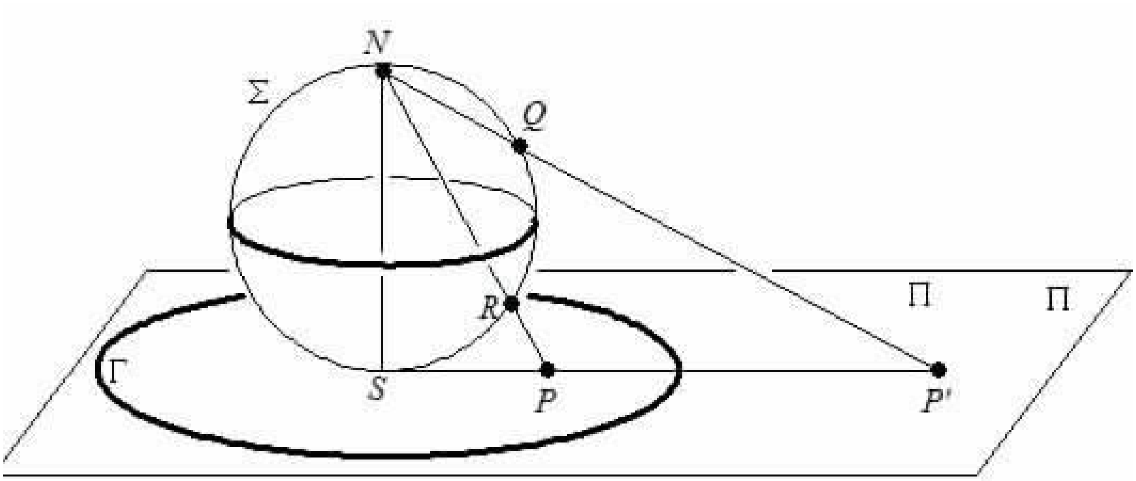
GEOMETRÍA COMPUTACIONAL

DEFORMACIÓN DE VARIEDADES DIFERENCIALES



LUCAS DE TORRE

Índice

1. Introducción 2

2. Material 2

3. Resultados 3

4. Conclusiones 3

5. Anexo A: Código 4

1. Introducción

Dada S_1^2 (2-esfera de radio unitario), buscamos estimar y representar una malla con 25 valores de latitud y 50 valores de longitud para, después, estimar y representar la imagen de la proyección estereográfica $\Pi : S_1^2 \setminus e_3 \to \mathbb{R}^2$. Finalmente, diseñaremos una curva sobre S_1^2 para comprobar cómo se deforma al proyectarla sobre $\mathbb{R}^2$.

Por otro lado, obtendremos una animación (de al menos 15 fotogramas) de la familia paramétrica:

$$f_t : S_1^2 \setminus e_3 \to \mathbb{R}^3 \tag{1}$$

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \to \frac{1}{(1-t)+|-1-z|t} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ (-1)t + z(1-t) \end{pmatrix} \tag{2}$$

2. Material

Partiendo de la plantilla proporcionada, para obtener la malla únicamente debemos cambiar las coordenadas polares (u y v) para que tomen los 25 valores de latitud y los 50 de longitud (con $u \in [0,\pi)$ y $v \in [0,2\pi)$), y después las transformaremos a coordenadas cartesianas. Después definimos la curva sobre S_1^2 variando de manera mínima la de la plantilla:

$$\begin{aligned} t &\in (0,1] \\ x(t) &= |t|\sin(107\frac{t}{2}) \\ y(t) &= |t|\cos(107\frac{t}{2}) \\ z(t) &= \sqrt{1-x^2-y^2} \end{aligned}$$

Ahora, después de definir la malla y la curva, las dibujamos con el comando *plot*. Por último, mediante la función *proj* proporcionada en la plantilla, estimamos la proyección estereográfica de la esfera y de nuestra curva para finalmente representarlas gráficamente.

Por otro lado, buscamos una animación de la familia paramétrica dada por (1) y (2), la cual transforma la esfera unidad en el plano $z = -1$. Para ello, primero escribimos una función (que llamamos *proj2*) para, dados t y z, realizar la transformación de f_t

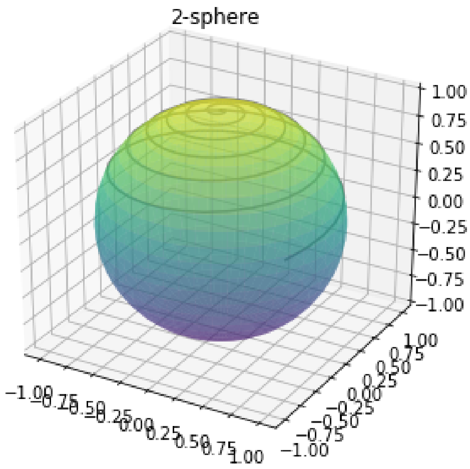
para x e y (la de z la hacemos directamente):

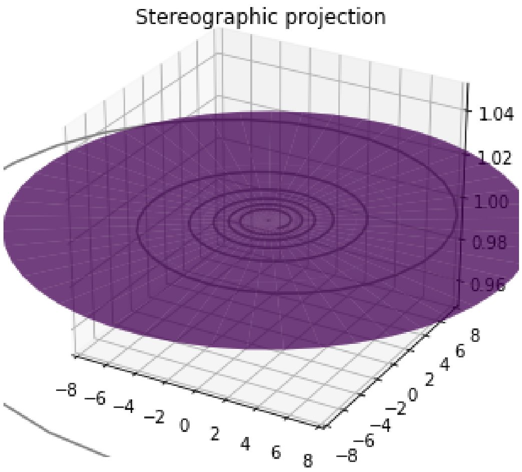
$$x(t) = \text{proj2}(x, z, t)$$
$$y(t) = \text{proj2}(y, z, t)$$
$$z(t) = -1 * t + z * (1 - t)$$

Después, utilizamos la función *animation.FuncAnimation* con 20 intervalos (más de 15), para, mediante el uso de la función proporcionada *animate*, hacer la representación gráfica de f_t sobre la esfera para 20 valores de t equiespaciados en [0,1] y, con esas representaciones gráficas, realizar la animación.

3. Resultados

La malla que hemos definido la podemos visualizar en las imágenes de debajo. La curva que hemos diseñado es la mencionada al comienzo del apartado anterior (la definida por t, $x(t)$, $y(t)$ y $z(t)$). Podemos ver la esfera y la curva sin proyectar a la izquierda y proyectadas a la derecha.





La animación dada por la familia paramétrica definida por (1) y (2) la podemos encontrar en el archivo *animacion.gif*

4. Conclusiones

Resulta muy interesante lo bien que se observa como, en función de cuánto se acerca una curva al polo norte de la esfera, la proyección de la curva "se aleja" de la propia esfera inicial.

5. Anexo A: Código

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 10 18:58:33 2020
@author: Jorge Sainero y Lucas de Torre
"""

#from mpl_toolkits import mplot3d

import os
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

os.getcwd()


"""
Ejemplo1

fig = plt.figure()
ax = plt.axes(projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
cset = ax.contour(X, Y, Z, 16, extend3d=True)
ax.clabel(cset, fontsize=9, inline=1)
plt.show()


Ejemplo2

def g(x, y):
    return np.sqrt(1-x ** 2 - y ** 2)
x = np.linspace(-1, 1, 30)
y = np.linspace(-1, 1, 30)
X, Y = np.meshgrid(x, y)
Z = g(X, Y)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 10, cmap='binary')
ax.contour3D(X, Y, -1*Z, 10, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')


Ejemplo3

fig = plt.figure()
```

```
ax = plt.axes(projection='3d')
t2 = np.linspace(1, 0, 100)
x2 = t2 * np.sin(20 * t2)
y2 = t2 * np.cos(20 * t2)
z2 = np.sqrt(1-x2**2-y2**2)
c2 = x2 + y2
ax.scatter(x2, y2, z2, c=c2)
ax.plot(x2, y2, z2, '-b')
2-sphere
"""

u = np.linspace(0, np.pi, 25)
v = np.linspace(0, 2 * np.pi, 50)

#Cambiamos de coordenadas polares a cartesianas
x = np.outer(np.sin(u), np.sin(v))
y = np.outer(np.sin(u), np.cos(v))
z = np.outer(np.cos(u), np.ones_like(v))

#Definimos una curva en la superficie de la esfera
t2 = np.linspace(0.001, 1, 200)
x2 = abs(t2) * np.sin(107 * t2/2)
y2 = abs(t2) * np.cos(107 * t2/2)
z2 = np.sqrt(1-x2**2-y2**2)

z0 = -1
def proj(x,z,z0=1,alpha=1):
    z0 = z*0+z0
    eps = 1e-16
    x_trans = x/(abs(z0-z)**alpha+eps)
    return(x_trans)
    #N tese que a adimos un  psilon  para evitar dividir entre
   0!!



def apartado1():
    print("Apartado 1:\n")
    global x,y,z,t2,x2,y2,z2


    """
    2-esfera y su proyecci n estereogr fica
    """

    fig = plt.figure(figsize=(12,12))
    fig.subplots_adjust(hspace=0.4, wspace=0.2)

    ax = fig.add_subplot(2, 2, 1, projection='3d')
```

```
    ax.plot_surface(x, y, z, rstride=1, cstride=1,alpha=0.5,
                    cmap='viridis', edgecolor='none')
    ax.plot(x2, y2, z2, '-b',c="gray")
    ax.set_title('2-sphere');
    ax = fig.add_subplot(2, 2, 2, projection='3d')
    ax.set_xlim3d(-8,8)
    ax.set_ylim3d(-8,8)
    ax.plot_surface(proj(x,z), proj(y,z), z*0+1, rstride=1, cstride=1, alpha=0.5,
                    cmap='viridis', edgecolor='none')
    ax.plot(proj(x2,z2), proj(y2,z2), 1, '-b',c="gray")
    ax.set_title('Stereographic projection');

    plt.show()
    plt.close(fig)

def proj2(x,z,t,z0=-1,alpha=1):
    z0 = z*0+z0
    eps = 1e-16
    x_trans = x/((1-t)+t*(abs(z0-z)**alpha+eps))
    return(x_trans)
    #N tese que a adimos un  psilon  para evitar dividir entre 0!!




from matplotlib import animation
    #from mpl_toolkits.mplot3d.axes3d import Axes3D

def animate(t):
    xt = proj2(x,z,t)
    yt = proj2(y,z,t)
    zt = -1*t + z*(1-t)
    x2t = proj2(x2,z2,t)
    y2t = proj2(y2,z2,t)
    z2t = -1*t + z2*(1-t)

    ax = plt.axes(projection='3d')
    ax.set_zlim3d(-1,1)
    ax.plot_surface(xt, yt, zt, rstride=1, cstride=1,alpha=0.5,
                    cmap='viridis', edgecolor='none')
    ax.plot(x2t,y2t, z2t, '-b',c="gray")
    return ax,

def init():
    return animate(0),
```

```
def apartado2():
    print("Apartado 2:\n")
    global x,y,z,t2,x2,y2,z2,z0

    t = 0.1


    xt = proj2(x,z,t)
    yt = proj2(y,z,t)
    zt = -1*t + z*(1-t)
    x2t = proj2(x2,z2,t)
    y2t = proj2(y2,z2,t)
    z2t = -1*t + z2*(1-t)

    fig = plt.figure(figsize=(6,6))
    ax = plt.axes(projection='3d')


    ax.set_xlim3d(-1,1)
    ax.set_ylim3d(-1,1)
    ax.plot_surface(xt, yt, zt, rstride=1, cstride=1,alpha=0.5,
                    cmap='viridis', edgecolor='none')
    ax.plot(x2t,y2t, z2t, '-b',c="gray")

    plt.show()
    plt.close(fig)

    """
    HACEMOS LA ANIMACI N
    """



    animate(np.arange(0, 1,0.1)[1])
    plt.show()


    fig = plt.figure(figsize=(12,12))
    ani = animation.FuncAnimation(fig, animate, np.arange(0,1,0.05)
    , init_func=init,
                                  interval=20)
    ani.save("animacion.gif", fps = 5)


apartado1()
apartado2()
```