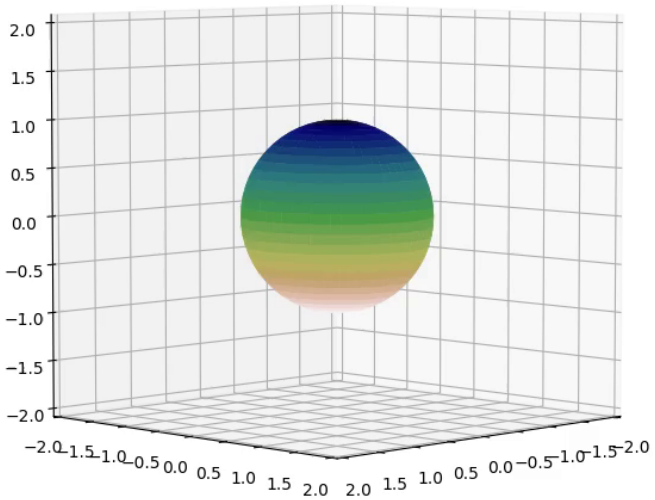
Práctica 5 - GCOMP

Celia Rubio Madrigal

7 de abril de 2022

Índice



*Para ver este gráfico en movimiento, se recomienda abrir este PDF en Adobe Reader u Okular.

1. Introducción

En esta práctica calcularemos y representaremos el difeomorfismo de una proyección estereográfica. Proyectaremos la superficie $S_1^2 \setminus \{e_3\}$ en $\mathbb{R}^2$, donde S_1^2 es la esfera bidimensional de radio unitario, y e_3 se corresponde al punto $(0, 0, 1)$, el polo norte de la esfera.

Verificaremos que esta "casi-esfera" $S_1^2 \setminus \{e_3\}$ es homeomorfa al plano, de la misma forma que, pudiendo generalizarse a más dimensiones, la superficie $S_1^n \setminus \{x_0\} \subset \mathbb{R}^{n+1}$ es homeomorfa al espacio euclidiano $\mathbb{R}^n$, mediante un homeomorfismo similar al que aquí presentamos.

2. Material usado y metodología

En primer lugar, debemos discretizar la esfera, una superficie continua, para trabajar con una malla de valores discretos que podamos manejar y representar con las herramientas de Python disponibles.

Para ello, tomamos un conjunto de 30 valores equidistanciados en el intervalo $[0, \pi)$, y 60 en el intervalo $[0, \tau)$, para representar valores de latitud, ϕ, y longitud, φ, respectivamente. La resolución de la rejilla, por tanto, es de aproximadamente $r_0 = 0.105 \pm 0.001$ radianes.

Para no incluir el punto e_3, hemos reducido el intervalo de latitudes a $[0.1, \pi)$. Desde la perspectiva continua, estamos eliminando todo un casquete y no un solo punto. Pero, desde la perspectiva discreta, de igual manera no podremos nunca representar la proyección en valores muy grandes.

Una vez obtenida la malla de coordenadas polares, mediante una transformación cartesiana encontramos los valores cartesianos equivalentes que conforman la casi-esfera. La transformación escogida, que se encuentra codificada en la función `get_sphere`, es:

$$\begin{aligned} x &= \sin(\phi) \times \sin(\varphi) \\ y &= \sin(\phi) \times \cos(\varphi) \\ z &= \cos(\phi) \times (1)_{i=1}^{|\varphi|} \end{aligned}$$

2.1. Apartado *i)*

En el primer apartado, representaremos la casi-esfera y una curva γ sobre ella. Después, proyectaremos ambos objetos con la proyección estereográfica Π_α, con $\alpha = 0.5$. Para $i = 3$, $\Pi_\alpha(x_i) = -1$; así estaremos embebiendo el plano $\mathbb{R}^2$ en el plano $\{z = -1\} \subset \mathbb{R}^3$. Para $i \in \{1, 2\}$:

$$\Pi_\alpha(x_i) = \frac{x_i}{(1 - x_3)^\alpha}$$

Finalmente, representaremos las proyecciones en la misma gráfica que sus contrapartes sin proyectar. La casi-esfera y su proyección se representan mediante la función `plot_sphere`, y la curva y su proyección mediante `plot_curve`.

La curva γ escogida, parametrizada con $t \in [0.5, 1]$ y con paso de rejilla $r_\gamma = 0.025$, es:

$$\begin{aligned} x &= t \cdot \sin(t^3) \\ y &= -t \cdot \cos(2 \cdot t^3) \\ z &= 1 - x^2 - y^2 \end{aligned}$$

2.2. Apartado *ii*)

En el segundo apartado, representaremos una sucesión de transformaciones del objeto $S_1^2 \setminus \{e_3\}$, tales que comiencen con la identidad y terminen con una proyección estereográfica.

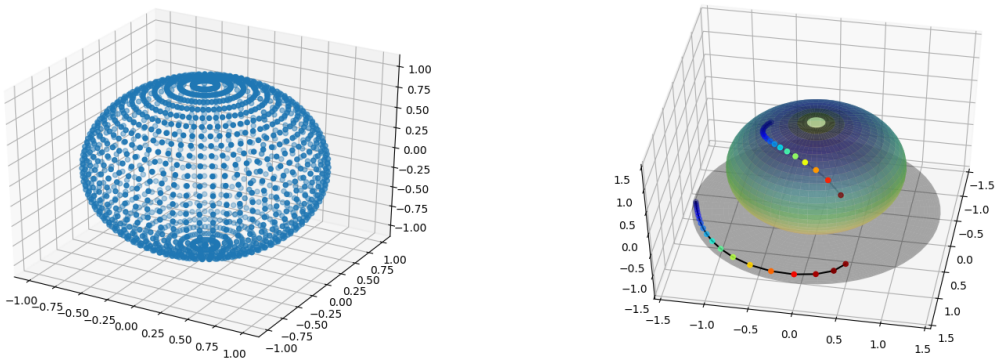
Tomaremos, para ello, la familia de funciones $f_t : S_1^2 \setminus \{e_3\} \mapsto \mathbb{R}^3$ dependientes de un parámetro temporal $t \in [0, 1]$:

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

3. Resultados y conclusiones

3.1. Apartado *i*)

A la izquierda se encuentra la malla de puntos que conforman la esfera, y a la derecha la gráfica de los objetos $S_1^2 \setminus \{e_3\}$, de $\Pi_\alpha\left(S_1^2 \setminus \{e_3\}\right)$, de γ y de $\Pi_\alpha(\gamma)$.



Para que la curva pudiera distinguirse encima de la esfera —que es debido a un problema interno de la herramienta gráfica de Python utilizada —hemos necesitado añadir un $\varepsilon = 0.1$ a la coordenada z de γ y de su proyección.

3.2. Apartado *ii*)

En los archivos adjuntos, así como en la portada de este trabajo, se encuentra la animación de la sucesión de funciones paramétricas f_t representadas a lo largo de t. En los últimos fotogramas se puede apreciar la ausencia del casquete superior de la esfera, como ya se ha indicado anteriormente.

Con estas representaciones visuales de proyecciones estereográficas, hemos confirmado que la esfera bidimensional a la que le falta un punto es homeomorfa al plano bidimensional. Hemos trabajado con cambios de coordenadas, y hemos descubierto cómo se deforma la esfera al aplicarle estas proyecciones.

4. Código

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d, Axes3D

""" Apartado 1 """


def get_sphere(n, m):
    u = np.linspace(0.1, np.pi, n)
    v = np.linspace(0, 2 * np.pi, m)
    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))
    return x, y, z


def plot_sphere(x, y, z, ax=None, lim=3, alpha=1, cmap="gist_earth_r"):
    if ax == None:
        fig = plt.figure()
        fig.tight_layout()
        ax = Axes3D(fig)
    ax.set_xlim3d(-lim, lim)
    ax.set_ylim3d(-lim, lim)
    ax.set_zlim3d(-lim, lim)
    ax.plot_surface(x, y, z, cmap=cmap, alpha=alpha, rstride=1, cstride=1)
    return ax


def plot_curve(x2, y2, z2, ax=None, lim=3):
    if ax == None:
        fig = plt.figure()
        fig.tight_layout()
        ax = Axes3D(fig)
    ax.set_xlim3d(-lim, lim)
    ax.set_ylim3d(-lim, lim)
    ax.set_zlim3d(-lim, lim)
    ax.plot(x2, y2, z2 + 0.1, "-b", c="black", zorder=3)
    ax.scatter(x2, y2, z2 + 0.1, c=x2 + y2, cmap="jet")
    return ax


t2 = np.linspace(0.5, 1, 20)
x2 = abs(t2) * np.sin(t2 ** 3)
y2 = -abs(t2) * np.cos(2 * t2 ** 3)
z2 = np.sqrt(1 - x2 ** 2 - y2 ** 2)

x, y, z = get_sphere(30, 60)
```

```
fig = plt.figure()
fig.tight_layout()
ax = Axes3D(fig)
ax.scatter(x, y, z)
plt.savefig('0')


ax = plot_sphere(x, y, z, alpha=0.66, lim=1.5)
plot_curve(x2, y2, z2, ax, lim=1 - 5)


def proj(x, y, z, alpha=1):
    eps = 1e-16
    aux = 1 / ((1 - z) ** alpha + eps)
    return aux * x, aux * y, 0 * z - 1


px, py, pz = proj(x, y, z, alpha=1 / 2)
px2, py2, pz2 = proj(x2, y2, z2, alpha=1 / 2)

plot_sphere(px, py, pz, ax, alpha=0.33, lim=1.5, cmap="gist_gray")
plot_curve(px2, py2, pz2, ax, lim=1.5)

ax.view_init(50, 10)
plt.savefig("1")


""" Apartado 2 """

from matplotlib import animation


def proj2(x, y, z, t):
    eps = 1e-16
    aux = 2 / (2 * (1 - t) + (1 - z) * t + eps)
    return aux * x, aux * y, -t + z * (1 - t)


def animate(t):
    xt, yt, zt = proj2(x, y, z, t)
    ax = Axes3D(fig)
    plot_sphere(xt, yt, zt, ax, alpha=1, lim=2)
    ax.view_init(5, 45)
    return (ax,)


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

```
fig = plt.figure(figsize=(6, 6))
fig.tight_layout()
ani = animation.FuncAnimation(
    fig, animate, np.linspace(0, 0.9, 20), init_func=init, interval=20
)
ani.save("2.mp4", fps=5)
```