Part I - Rotation

Problem 1: $p = \frac{i+1}{\sqrt{2}}$ $\quad q = \frac{j+1}{\sqrt{2}}$

unit norm quaternion.

① $\frac{p+q}{2} = \frac{1}{\sqrt{2}} + \frac{i}{2\sqrt{2}} + \frac{j}{2\sqrt{2}}$

$= \left[\frac{1}{\sqrt{2}}, \frac{1}{2\sqrt{2}}(i+j)\right]$

$\left|\frac{p+q}{2}\right| = \sqrt{\frac{1}{2} + \frac{1}{8} + \frac{1}{8}} = \frac{\sqrt{3}}{2}$

Quaternion $r = \frac{\frac{p+q}{2}}{\left|\frac{p+q}{2}\right|} = \frac{2}{\sqrt{3}}\left[\frac{1}{\sqrt{2}}, \frac{1}{2\sqrt{2}}(i+j)\right]$

$r = \frac{\sqrt{2}}{\sqrt{3}} + \frac{i}{\sqrt{6}} + \frac{j}{\sqrt{6}}, \quad |r| = 1$

Calculating rotation matrix $M(r)$

Let $M(r) = Rot(\hat{\omega}, \theta)$

then, $\cos\frac{\theta}{2} = \frac{\sqrt{2}}{\sqrt{3}}$, $\sin\left(\frac{\theta}{2}\right)\hat{\omega} = \frac{1}{\sqrt{6}}(i+j)$

$$\Rightarrow \boxed{\theta = 2\cos^{-1}\left(\frac{\sqrt{2}}{\sqrt{3}}\right)}, \quad \sin\left(\frac{\theta}{2}\right)\hat{\omega} = \frac{1}{\sqrt{3}}\left[\frac{1}{\sqrt{2}}i + \frac{1}{\sqrt{2}}j\right]$$

$$\sin\frac{\theta}{2} = \frac{1}{\sqrt{3}}$$

Axis of rotation of $M(r)$

$$\boxed{\hat{\omega} = \frac{1}{\sqrt{2}}i + \frac{1}{\sqrt{2}}j}$$

$$\hat{\omega} = \frac{1}{\sqrt{2}}(i+j)$$

Angle of rotation $\theta = 2\cos^{-1}\left(\frac{\sqrt{2}}{\sqrt{3}}\right) \simeq 70.53^\circ$

$$M(r) = e^{[\hat{\omega}]\theta}$$

$$= I + [\hat{\omega}]\sin\theta + [\hat{\omega}]^2(1-\cos\theta)$$

$$[\hat{\omega}] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1/\sqrt{2} \\ 0 & 0 & -1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \end{bmatrix}$$

$$= \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$[\hat{\omega}]^2 = \frac{1}{2} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

$$= \frac{1}{2} \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

Rotation Matrix

$$M(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \frac{2}{3} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} + \frac{1}{3} \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

$$= \begin{bmatrix} 2/3 & 1/3 & 2/3 \\ 1/3 & 2/3 & -2/3 \\ -2/3 & 2/3 & 1/3 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & -2 \\ -2 & 2 & 1 \end{bmatrix}$$

(2) $$p = \frac{1+i}{\sqrt{2}} = \left[\frac{1}{\sqrt{2}}, \frac{i}{\sqrt{2}}\right]$$

Let exponential coordinate of $p = \hat{\omega}_p \theta_p = \vec{\omega}_p$

axis of rotation

angle of rotation

$$\cos\left(\frac{\theta_p}{2}\right) = \frac{1}{\sqrt{2}}$$

$$\sin\left(\frac{\theta_p}{2}\right)\hat{\omega}_p = \frac{i}{\sqrt{2}}$$

$\theta_p = \pi/2$ radians

$\theta_p = 90$ degrees

$$\sin\frac{\theta_p}{2} = \frac{1}{\sqrt{2}}, \quad \hat{\omega}_p = i$$

$\theta_p = \pi/2$ radians

$\hat{\omega}_p = i$

$$\vec{\omega}_p = \hat{\omega}_p \theta_p$$

$$\vec{\omega}_p = \frac{\pi}{2}\begin{bmatrix}1\\0\\0\end{bmatrix}$$

Let exponential coordinate of $q = \vec{\omega}_q = \hat{\omega}_q \theta_q$

$$q = \left[\frac{1}{\sqrt{2}} \cdot j/\sqrt{2} \right]$$

$$\cos \frac{\theta_q}{2} = \frac{1}{\sqrt{2}} \qquad \sin\left(\frac{\theta_q}{2}\right) \hat{\omega}_q = \frac{1}{\sqrt{2}} j$$

$$\theta_q = \frac{\pi}{2} \text{ radians} \qquad \hat{\omega}_q = j = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\theta_q = 90 \text{ degrees}$$

$$\vec{\omega}_q = \hat{\omega}_q \theta_q = \frac{\pi}{2} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

③ Skew symmetric repr. of rotation.

$$\omega_p = \frac{\pi}{2}\begin{bmatrix}1\\0\\0\end{bmatrix} \qquad \omega_q = \frac{\pi}{2}\begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$[\omega_p] = \frac{\pi}{2}\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{bmatrix}$$

$$[\omega_q] = \frac{\pi}{2}\begin{bmatrix}0 & 0 & 1\\ 0 & 0 & 0\\ -1 & 0 & 0\end{bmatrix}$$

$$R_p = M(p) = Rot(\hat{\omega}_p, \theta_p) = \exp^{[\hat{\omega}_p]\theta_p}$$

$$= I + [\hat{\omega}_p]\sin\theta_p + [\hat{\omega}_p]^2(1-\cos\theta_p)$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & -1\end{bmatrix}$$

$$M(p) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \det(M(p)) = 1$$

$$M(p)^T M(p) = I$$

$$R_q = M(q) = Rot(\hat{\omega}_q, \theta_q) = \exp^{[\hat{\omega}_q]\theta_q}$$

$$= I + [\hat{\omega}_q] \sin\theta_q + [\hat{\omega}_q]^2 (1 - \cos\theta_q)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$M(q) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \qquad \det(M(q)) = 1$$

$$M(q)^T M(q) = I$$

(b) To verify: $\exp^{([\omega_1]+[\omega_2])} \stackrel{?}{=} \exp^{[\omega_1]} \cdot \exp^{[\omega_2]}$

$$\text{LHS} = \exp\left([\omega_1] + [\omega_2]\right)$$

Let $\omega_1 = \omega_p$ and $\omega_2 = \omega_q$.

then

$$\exp\left([\omega_1] + [\omega_2]\right) = \exp\left([\omega_p] + [\omega_q]\right)$$

$$= \exp\left(\underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}}_{\hat{\omega}} \right)$$

$$= I + [\hat{\omega}]\sin(1) + [\hat{\omega}]^2(1 - \cos(1))$$

$$[\hat{\omega}]^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

$$\text{RHS} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}\sin(1) + \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & -2 \end{bmatrix}(1 - \cos(1))$$

$$LHS = \begin{bmatrix} \cos(1) & 1-\cos(1) & \sin(1) \\ 1-\cos(1) & \cos(1) & -\sin(1) \\ -\sin(1) & \sin(1) & -1+2\cos(1) \end{bmatrix}$$

Now,

$$RHS = \exp\left([\omega_1]\right)\exp\left([\omega_2]\right) = \exp\left([\omega_p]\right)\exp\left([\omega_q]\right)$$

$$\exp\left([\omega_p]\right) = I + [\omega_p]\sin(1) + [\omega_p]^2(1-\cos 1)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}\sin(1) + \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}(1-\cos 1)$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(1) & -\sin(1) \\ 0 & \sin(1) & \cos(1) \end{bmatrix}$$

$$\exp\left([\omega_q]\right) = I + [\omega_q]\sin(1) + (1-\cos(1))[\omega_q]^2$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \sin(1) + \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} (1-\cos 1)$$

$$= \begin{bmatrix} \cos(1) & 0 & \sin(1) \\ 0 & 1 & 0 \\ -\sin(1) & 0 & \cos(1) \end{bmatrix}$$

$$RHS = \exp([\omega_p]) \exp([\omega_q])$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 1 & -\sin 1 \\ 0 & \sin 1 & \cos 1 \end{bmatrix} \begin{bmatrix} \cos 1 & 0 & \sin 1 \\ 0 & 1 & 0 \\ -\sin 1 & 0 & \cos 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 1 & 0 & \sin 1 \\ \sin^2 1 & \cos 1 & -\sin 1 \cos 1 \\ -\sin 1 \cos 1 & \sin(1) & \cos^2 1 \end{bmatrix}$$

Thus $\text{LHS} \neq \text{RHS}$ for $\omega_1 = \omega_p$ & $\omega_2 = \omega_q$

Hence in general, the following relationship does not hold for exponential map

$$\exp\left([\omega_1] + [\omega_2]\right) = \exp\left([\omega_1]\right) \exp\left([\omega_2]\right)$$

(c) Two point clouds $X, Y \in \mathbb{R}^{3\times n}$

Original point cloud alignment problem.

$$\underset{R}{\text{minimize}} \quad \|RX - Y\|_F^2$$

$$\text{subject to} \quad R^T R = I \;\&\; \det(R) = 1$$

Now, given R_1, rotation matrices in local neighbourhood of R_1 can be parametrized as

$$R_2 = R_1 \exp([\Delta\omega]) \simeq R_1 \left(I + [\Delta\omega]\right) \text{ for } \Delta\omega \approx 0.$$

Routine to find $\Delta\omega$:

$$\|RX - Y\|_F^2 \approx \|R_1 e^{[\Delta\omega]} X - Y\|_F^2$$

$$= \|R_1 (I + [\Delta\omega]) X - Y\|_F^2$$

$$= \|R_1 [\Delta\omega] X - (Y - R_1 X)\|_F^2$$

Let $(Y - R_1 X) = C \in \mathbb{R}^{3xn}$

$$R_1 [\Delta\omega] X = R_1 \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & & | \end{bmatrix}}_{X}$$

Where $\Delta\omega = [\omega_1, \omega_2, \omega_3]^T$

Using $[\Delta\omega] x_i = (\Delta\omega \times x_i)$

$$R_1 [\Delta\omega] X = R_1 \begin{bmatrix} | & | & & | \\ \Delta\omega \times x_1 & \Delta\omega \times x_2 & \cdots & \Delta\omega \times x_n \\ | & | & & | \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} - r_1^T - \\ - r_2^T - \\ - r_3^T - \end{bmatrix}}_{R} \underbrace{\begin{bmatrix} | & | & & | \\ \Delta\omega \times x_1 & \Delta\omega \times x_2 & \cdots & \Delta\omega \times x_n \\ | & | & & | \end{bmatrix}}_{[\Delta\omega] X}$$

$3x3$ $\quad$ $3xn$

$$= \begin{bmatrix} r_1^T(\Delta\omega \times x_1) & r_1^T(\Delta\omega \times x_2) & \cdots & r_1^T(\Delta\omega \times x_n) \\ r_2^T(\Delta\omega \times x_1) & r_2^T(\Delta\omega \times x_2) & \cdots & r_2^T(\Delta\omega \times x_n) \\ r_3^T(\Delta\omega \times x_1) & r_3^T(\Delta\omega \times x_2) & \cdots & r_3^T(\Delta\omega \times x_n) \end{bmatrix}$$

$$= \begin{bmatrix} (n_1 \times r_1^T)\cdot\Delta\omega & (n_2 \times r_1^T)\cdot\Delta\omega & \cdots & (n_n \times r_1^T)\Delta\omega \\ (n_1 \times r_2^T)\cdot\Delta\omega & (n_2 \times r_2^T)\cdot\Delta\omega & \cdots & (n_n \times r_2^T)\Delta\omega \\ (n_1 \times r_3^T)\cdot\Delta\omega & (n_3 \times r_3^T)\cdot\Delta\omega & \cdots & (n_n \times r_3^T)\Delta\omega \end{bmatrix}$$

Transforming the above matrix $R_1[\Delta\omega]X$ into $A\,\Delta\omega$. & reshaping the matrix $C \in \mathbb{R}^{3\times n}$ to $B \in \mathbb{R}^{3n\times 1}$

$$A\Delta\omega = \underbrace{\begin{bmatrix} (n_1 \times r_1^T) \\ n_1 \times r_2^T \\ n_1 \times r_3^T \\ n_2 \times r_1^T \\ n_2 \times r_2^T \\ n_2 \times r_3^T \\ \vdots \\ n_n \times r_1^T \\ n_n \times r_2^T \\ n_n \times r_3^T \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}}_{\Delta\omega} \quad 3n\times 3$$

Thus, $\| RX - Y \|_F^2 = \| A \Delta\omega - B \|_F^2$

→ In the above transformation process, we assumed to be given a rotation matrix $R_1 \in SO(3)$ and updating R_1 by a very small rotation $\Delta\omega \in \mathbb{R}^3$ using an approximation given by

$$R_2 = R_1 \exp([\Delta\omega])$$

$$= R_1 \left[I + [\Delta\omega] \sin(1) + [\Delta\omega]^2 (1 - \cos 1) \right]$$

$$R_2 \simeq R_1 (I + [\Delta\omega]) \quad \text{for } \Delta\omega \simeq 0$$

Thus modified optimization problem –

Linear Least square problem.

$$\underset{\Delta\omega}{\text{minimize}} \quad \| A \Delta\omega - B \|_F^2 \quad \text{s.t}$$

$$\Delta\omega^T \Delta\omega \leq \varepsilon$$

The above optimization problem is same as the one we solved in HW0 problem 1. Please check my HW0 solution for detailed solution.

(4) Double covering of quaternions

(a) $p' = -p = -\frac{(1+i)}{\sqrt{2}} = \left[-\frac{1}{\sqrt{2}}, -\frac{i}{\sqrt{2}}\right]$

$q' = -q = -\frac{(1+j)}{\sqrt{2}} = \left[-\frac{1}{\sqrt{2}}, -\frac{j}{\sqrt{2}}\right]$

$\cos\left(\frac{\theta_{p'}}{2}\right) = -\frac{1}{\sqrt{2}} \qquad \sin\left(\frac{\theta_{p'}}{2}\right)\hat{\omega}_{p'} = -\frac{i}{\sqrt{2}}$

$\theta_{p'} = 2\arccos\left(-\frac{1}{\sqrt{2}}\right) = 3\pi/2$ radians

$= 270$ degrees

$\sin\left(\frac{\theta_{p'}}{2}\right)\hat{\omega}_{p'} = -\frac{i}{\sqrt{2}} \Rightarrow \hat{\omega}_{p'} = -i$

Exponential coordinate of $p' = \theta_{p'}\,\hat{\omega}_{p'}$

$= -2\arccos\left(-\frac{1}{\sqrt{2}}\right)\begin{bmatrix}1\\0\\0\end{bmatrix}$

$= -\frac{3\pi}{2}\begin{bmatrix}1\\0\\0\end{bmatrix}$

Now, $q' = -q = -\frac{(1+j)}{\sqrt{2}}$

In a similar manner as shown above,

Exponential coordinate of $q' = \theta_{q'} \hat{\omega}_{q'}$

$$= -2\arccos\left(-\frac{1}{\sqrt{2}}\right) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$= -\frac{3\pi}{2} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

→ p : $\hat{\omega}_p = i$ $\quad \theta_p = \pi/2$

p' : $\hat{\omega}_{p'} = -i$ $\quad \theta_{p'} = 3\pi/2$

Thus, p & p' rotate any random vector $v \in \mathbb{R}^3$ to the same transformed vector $v' \in \mathbb{R}^3$.

→ Similarly for q: $\hat{\omega}_q = j$, $\theta_q = \pi/2$

q': $\hat{\omega}_{q'} = -j$, $\theta_{q'} = 3\pi/2$

q & q' rotate any random vector $v \in \mathbb{R}^3$ to the same transformed vector $v' \in \mathbb{R}^3$.

Hence, we hypothesize that p & p' have the same rotation matrix. Same condition applies for q & q'.

Now, we prove the above statement for a general quaternion pair $(r, -r)$

Let $r = [s, \vec{v}]$ & $r' = -r = [-s, -\vec{v}]$

$\cos\left(\frac{\theta_r}{2}\right) = s$ $\qquad$ $\cos\left(\frac{\theta_{r'}}{2}\right) = -s$

$\theta_r = 2\arccos(s)$ $\qquad$ $\theta_{r'} = 2\arccos(-s)$

$\sin\left(\frac{\theta_r}{2}\right)\hat{\omega}_r = \vec{v} = \lambda\hat{v}$ $\qquad$ $\sin\left(\frac{\theta_{r'}}{2}\right)\hat{\omega}_{r'} = -\vec{v} = -\lambda\hat{v}$

$\hat{\omega}_r = \frac{\vec{v}}{\sin\left(\frac{\theta_r}{2}\right)}$ $\qquad$ $\hat{\omega}_{r'} = \frac{-\vec{v}}{\sin\left(\frac{\theta_{r'}}{2}\right)}$

$$M(r) = I + [\hat{\omega}_r] \sin\theta_r + [\hat{\omega}_r]^2 (1-\cos\theta_r)$$

$$= I + \frac{[\vec{v}]}{\sin\left(\frac{\theta_r}{2}\right)} 2\sin\left(\frac{\theta_r}{2}\right)\cos\left(\frac{\theta_r}{2}\right) + \frac{[\vec{v}]^2}{\sin^2\left(\frac{\theta_r}{2}\right)} 2\sin^2\frac{\theta_r}{2}$$

$$= I + 2\cos\left(\frac{\theta_r}{2}\right)[\vec{v}] + 2[\vec{v}]^2$$

$$= I + 2s[\vec{v}] + 2[\vec{v}]^2$$

$$M(r') = I + [\hat{\omega}_{r'}] \sin\theta_{r'} + [\hat{\omega}_{r'}]^2 (1-\cos\theta_{r'})$$

$$= I + \frac{[-\vec{v}]}{\sin\frac{\theta_{r'}}{2}} 2\sin\frac{\theta_{r'}}{2}\cos\frac{\theta_{r'}}{2} + \frac{[-\vec{v}]^2}{\sin^2\frac{\theta_{r'}}{2}} 2\sin^2\frac{\theta_{r'}}{2}$$

$$= I - [\vec{v}] 2\cos\left(\frac{\theta_{r'}}{2}\right) + 2[\vec{v}]^2$$

$$= I + 2s[\vec{v}] + 2[\vec{v}]^2 = M(r$$

(Proved)

Thus for any two quaternion pair $(r, -r)$ the rotation matrices $M(r)$, $M(-r)$ are the same. Hence, any vector in $\mathbb{R}^3$ has the same transformation when rotated using r and $(-r)$.

(4) (b) No, we cannot use the L_2 distance between GT quaternion & predicted quaternion in a neural network regression.

Reason: Since quaternion $r = [s, \vec{v}]$ & $r' = -r = [-s, -\vec{v}]$ produce the same rotation matrix $M(r) = M(r')$ and therefore has the same transformed output.

Let $p \in \mathbb{R}^3$ be a vector, $P = [0, \vec{p}]$

then $p_r = r P r^{-1} = p_{r'} = r' P r'^{-1}$

vector p rotated by r & r' to produce p_r & $p_{r'}$ (resp.) and $p_r = p_{r'}$

Thus, for example, if we input a rotated image in a neural network to predict the rotation wrt a reference unrotated image, the predicted output can be r or $-r$ (which are both correct). But, if the GT is r, then in one case the loss is 0 while in other case the loss would be $\|2r\|_2^2$.

problem1

October 23, 2022

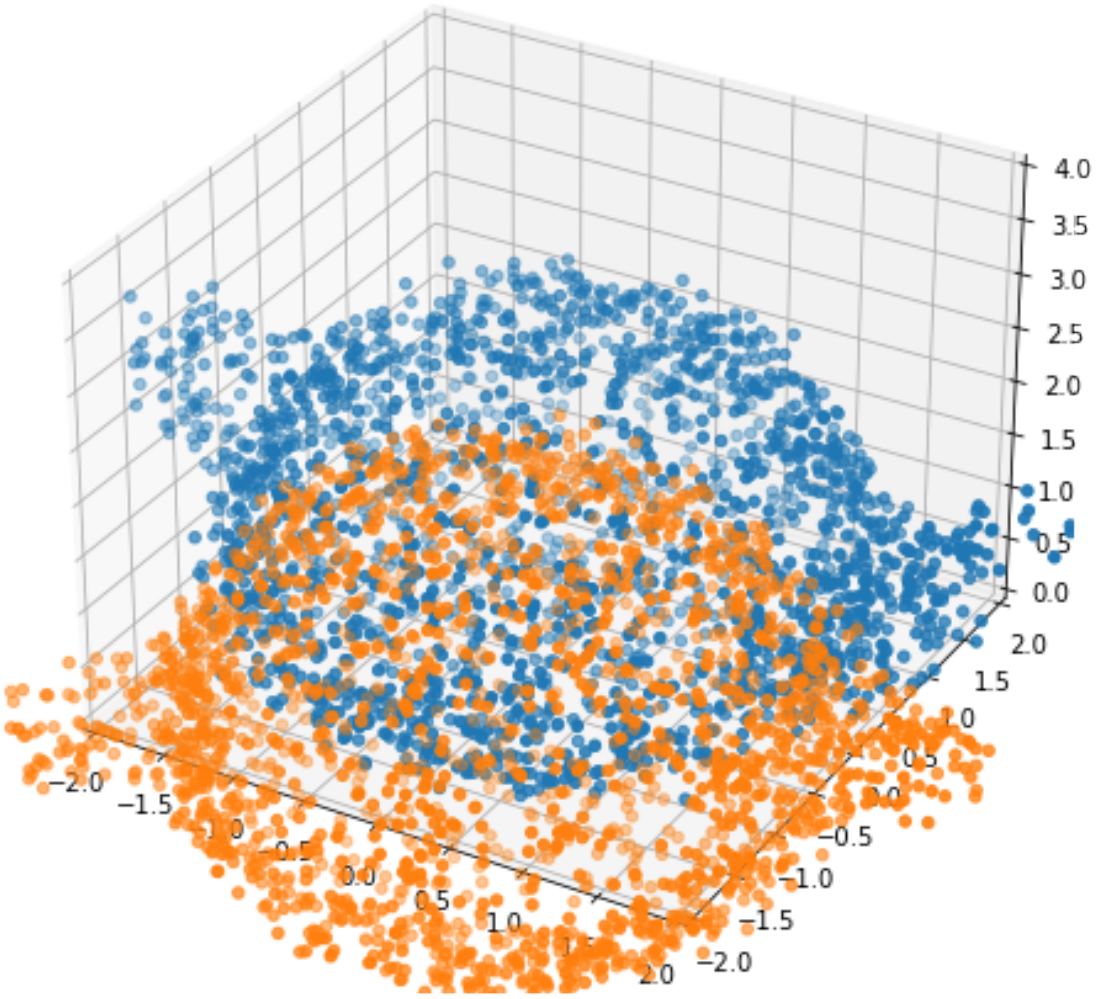
1 Part 1.3: Point Cloud Alignment Problem

[1]:
```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
```

[2]:
```
# Note Matplotlib is only suitable for simple 3D visualization.
# For later problems, you should not use Matplotlib to do the plotting
def show_points(points):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points[0], points[2], points[1])

def compare_points(points1, points2):
    fig = plt.figure(figsize=(12,8))
    ax = fig.gca(projection='3d')
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points1[0], points1[2], points1[1])
    ax.scatter(points2[0], points2[2], points2[1])
```

[3]:
```
npz = np.load('../data/HW1_P1.npz')
X = npz['X']
Y = npz['Y']

print(X.shape, Y.shape)
compare_points(X, Y)  # noisy teapotsand
```

```
(3, 2000) (3, 2000)
```



[4]:
```
def compute_hlambda(A, b, eps, lambd):
    term1 = np.matmul(A.T, A) + 2*lambd*np.eye(A.shape[1])
    term2 = np.linalg.inv(term1)
    term3 = np.matmul(A.T, b)

    hlambda = np.matmul(term2, term3)
    return hlambda
```

[5]:
```
def compute_glambda(A, b, eps, lambd):
    hlambda = compute_hlambda(A, b, eps, lambd)
    glambda = np.matmul(hlambda.T, hlambda) - eps

    return glambda
```

[6]:
```
def get_skew_from_vector(v):
    v = np.reshape(v, (v.shape[0],))
    return np.array([[0, -v[2], v[1]],
                     [v[2], 0, -v[0]],
                     [-v[1], v[0], 0]])
```

[7]:
```
# copy-paste your hw0 solve module here
def hw0_solve(A, b, eps):
    # Line search using Bisection method
    start = 0
    end = 10000000

    # your implementation here
    g_start = compute_glambda(A, b, eps, start)
    g_end = compute_glambda(A, b, eps, end)

#     print(g_start, g_end)

    if (g_start * g_end >= 0):
        print("You have not assumed right start and end points \n")
        return np.zeros(A.shape[1])

    mid = start
    while ((end-start) >= 1e-3):

        # Find middle point
        mid = (start+end)/2

        glambda_mid = compute_glambda(A, b, eps, mid)

        # Check if middle point is root
        if (glambda_mid <= 1e-14):
            break

        # Decide the side to repeat the steps
        if (glambda_mid * compute_glambda(A, b, eps, start) < 0):
            end = mid
        else:
            start = mid

#     print("Optimal lambda : ","%.4f"%mid)

    return compute_hlambda(A, b, eps, mid)
```

[13]:
```
R1 = np.eye(3)  # (3,3)
num_pts = X.shape[1]  # =n
assert(X.shape == Y.shape)
```

```
A = np.zeros((3*num_pts, 3))
B = np.zeros((3*num_pts, 1))
eps = 1e-5

# solve this problem here, and store your final results in R1
for __ in tqdm(range(10000)):

    B = Y - np.matmul(R1, X)   # (3,n)
    B = np.reshape(B, newshape=(3*num_pts,1), order='F')   # (3n,1)

#     for i in range(num_pts):
#         A[3*i+0,:] = np.cross(X[:,i], R1[0,:])
#         A[3*i+1,:] = np.cross(X[:,i], R1[1,:])
#         A[3*i+2,:] = np.cross(X[:,i], R1[2,:])

    temp1 = np.repeat(np.transpose(X), repeats=3, axis=0)
    temp2 = np.tile(R1, reps=(num_pts,1))
    A = np.cross(temp1, temp2)

    delta_omega = hw0_solve(A, B, eps)
#     print("Delta Omega:", delta_omega)

    # update rotation matrix R
    R1 = np.matmul(R1, (np.eye(3) + get_skew_from_vector(delta_omega)))
```

```
  0%|          | 0/10000 [00:00<?, ?it/s]
```

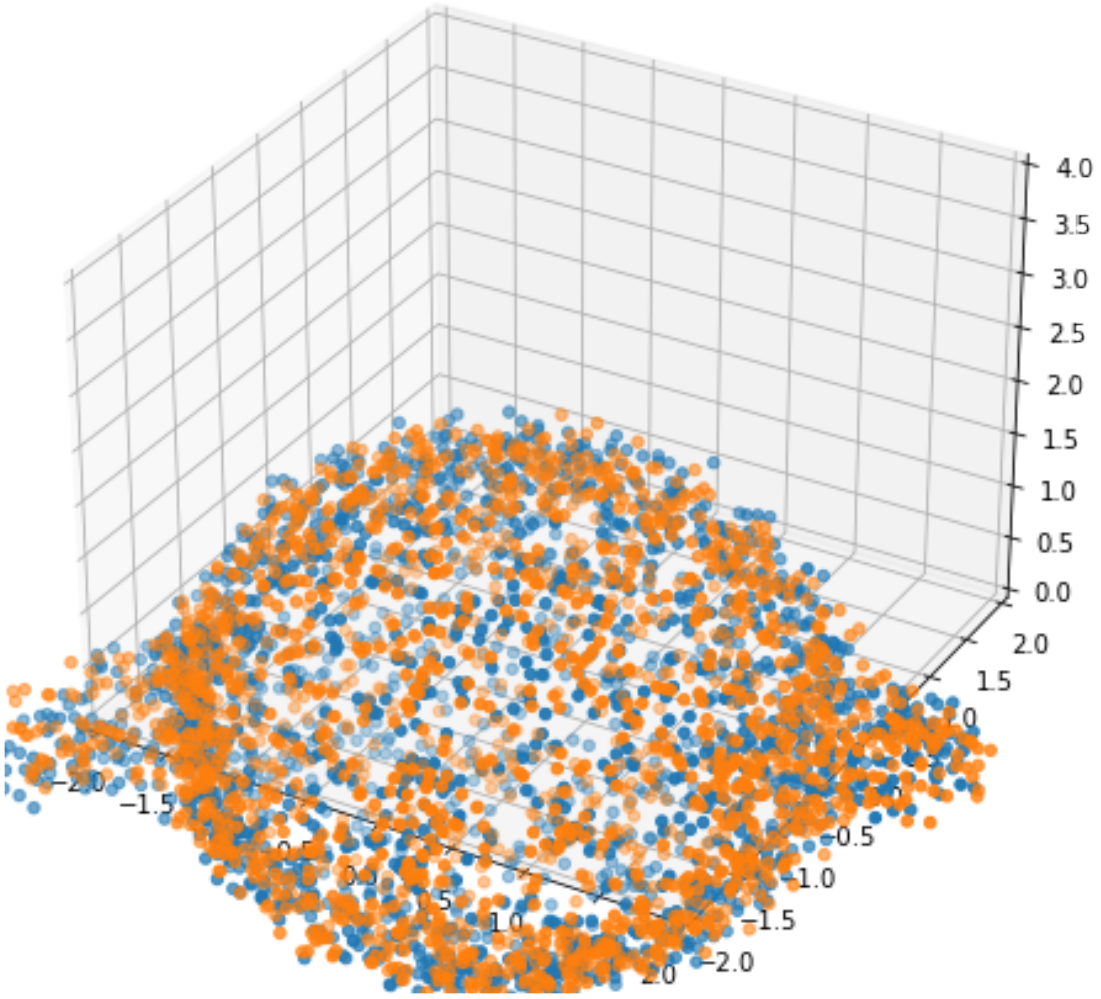
[9]:
```
# X_temp = np.random.randint(1, 5, (3,5))
# R_temp = np.random.randint(1, 5, (3,3))
# print(X_temp)
# print(R_temp)
```

[10]:
```
# X_temp = np.transpose(X_temp)
# # print(X_temp)
# X_temp = np.repeat(X_temp, 3, axis=0)
# print(X_temp)
# R_temp = np.tile(R_temp, (5,1))
# print(R_temp)
# np.cross(R_temp, X_temp)
# np.reshape(X_temp, (15,1), order='F')
```

1.1 Point cloud aligned result

[14]:
```
# Testing code, you should see the points of the 2 teapots roughly overlap
compare_points(R1@X, Y)
print("R.T R:\n", R1.T@R1, "\n")
print("det(R):", np.linalg.det(R1))
```

```
R.T R:
 [[ 1.00126409e+00 -2.20266262e-04  1.60448760e-04]
 [-2.20266262e-04  1.00042413e+00  5.19202708e-04]
 [ 1.60448760e-04  5.19202708e-04  1.00104319e+00]]

det(R): 1.0013657494101809
```



Part II: Differential Geometry

Problem 2:

$$f(u,v) = [a\cos u \sin v,\ b \sin u \sin v,\ c \cos v]$$

and $f: R^2 \to R^3$ $\quad -\pi < u < \pi$, $\quad 0 < v < \pi$

Given $a = 1,\ b = 1,\ c = \frac{1}{2}$

$p = (u, v) \in R^2$ is a point in domain of f.

$\gamma: (-1, 1) \to R^2$ is a curve with

$$\gamma(0) = p \qquad \gamma'(t) = v$$

$$\gamma(0) = \begin{bmatrix} \gamma_1(0) \\ \gamma_2(0) \end{bmatrix} = p \ \& \ \gamma'(t) = \begin{bmatrix} \delta\gamma_1(t)/\delta t \\ \delta\gamma_2(t)/\delta t \end{bmatrix}$$

$$= v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\boxed{\gamma(t) = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} t + p = vt + p}$$

$$f(\gamma(t)) = f(\gamma_1(t), \gamma_2(t))$$

$$= \begin{bmatrix} a\cos(\gamma_1(t)) \sin(\gamma_2(t)) \\ b\sin(\gamma_1(t)) \sin(\gamma_2(t)) \\ c\cos(\gamma_2(t)) \end{bmatrix}$$

$$f'(\gamma(t)) = (f \circ \gamma)'(t) = \frac{d}{dt} \begin{bmatrix} a\cos\gamma_1(t) \sin\gamma_2(t) \\ b\sin\gamma_1(t) \sin\gamma_2(t) \\ c\cos\gamma_2(t) \end{bmatrix}$$

$$\gamma(t) = \begin{bmatrix} \gamma_1(t) \\ \gamma_2(t) \end{bmatrix} = \begin{bmatrix} v_1 t + p_1 \\ v_2 t + p_2 \end{bmatrix}$$

$$(f \circ \gamma)'(t) = \frac{d}{dt} \begin{bmatrix} a\cos(v_1 t + p_1) \sin(v_2 t + p_2) \\ b\sin(v_1 t + p_1) \sin(v_2 t + p_2) \\ c\cos(v_2 t + p_2) \end{bmatrix}$$

$$(f \circ \gamma)'(t) = \begin{bmatrix} -aw_1 \sin(\gamma_1(t)) \sin(\gamma_2(t)) + aw_2 \cos(\gamma_1(t)) \cos(\gamma_2(t)), \\ bv_1 \cos(\gamma_1(t)) \sin(\gamma_2(t)) + bv_2 \sin(\gamma_1(t)) \cos(\gamma_2(t)), \\ -cv_2 \sin(\gamma_2(t)) \end{bmatrix}$$

① Let $p = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ be a point in the domain of f.

Let $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $\quad \gamma(t) = vt + p$

$$= \begin{bmatrix} t+1 \\ t+1 \end{bmatrix} \in \text{dom}(f) \ \forall \ t \in (-1, 1)$$

$$(f \circ \gamma)(t) = \begin{bmatrix} \cos(t+1) \sin(t+1) \\ \sin(t+1) \sin(t+1) \\ \frac{1}{2} \cos(t+1) \end{bmatrix}$$

Since $t \in (-1, 1)$, $(t+1) \in (0, 2)$

$(t+1, t+1)$ lies in the domain of f

$$(f \circ \gamma)'(0) = \begin{bmatrix} \cos^2(t+1) - \sin^2(t+1) \\ 2\sin(t+1)\cos(t+1) \\ -\frac{1}{2}\sin(t+1) \end{bmatrix}_{t=0}$$

$$(f \circ \gamma)'(0) = \begin{bmatrix} \cos(2) \\ \sin(2) \\ -\frac{1}{2}\sin(1) \end{bmatrix} \approx \begin{bmatrix} -0.416 \\ 0.909 \\ -0.421 \end{bmatrix}$$

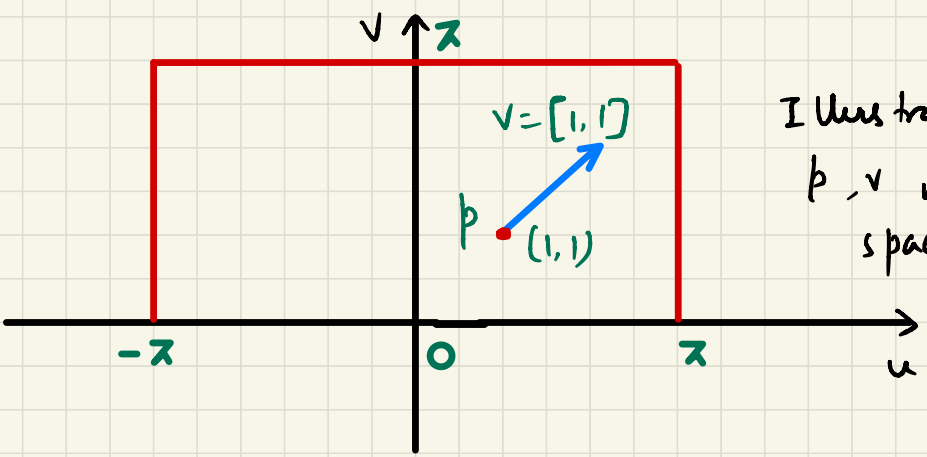


Illustration of p, v in u-v space

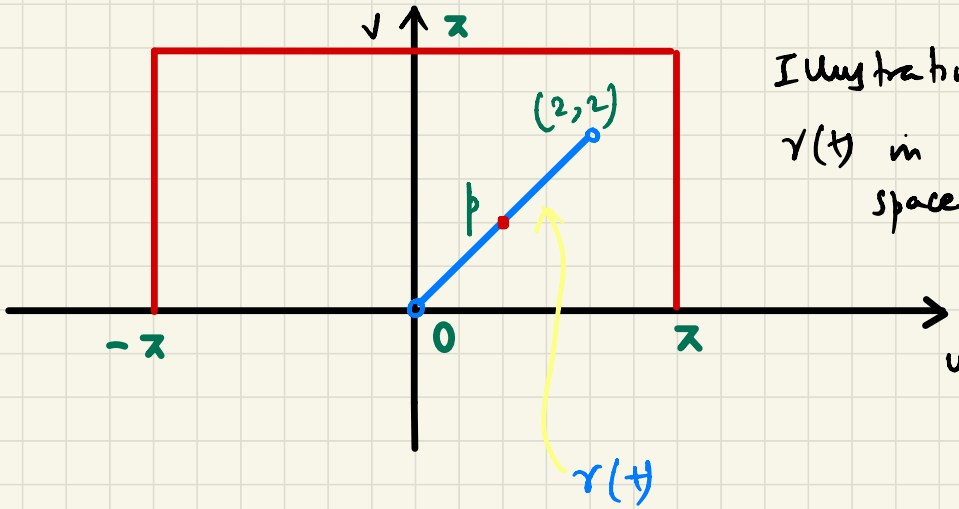
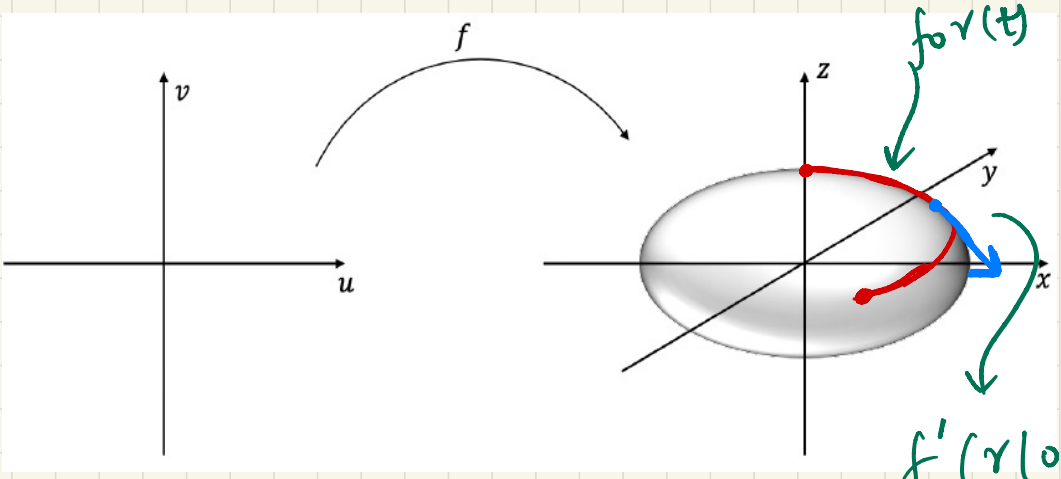


Illustration of $r(t)$ in u-v space



At $t=-1$, $f \circ r(t) = [0,\ 0,\ \tfrac{1}{2}]$

At $t=+1$, $f \circ r(t) = [-0.38,\ 0.83,\ -0.21]$

At $t=0$, $f \circ r(t) = [0.45,\ 0.71,\ 0.27]$

$f'(r(0)) = \begin{bmatrix} -0.416 \\ 0.909 \\ -0.44 \end{bmatrix}$

(*) $f(r(t))$ & $f'(r(0))$ shown in code (plot) as well.

Problem 2

(3) Differential of a function f at a point p defined by the gradient of curve wrt curve parameter

$$Df_p(v) = (f \circ \gamma)'(t)\Big|_{t=0}$$

(a) $$Df_p = \left[\partial f/\partial u \;.\; \partial f/\partial v\right]_{3\times 2}$$

$$Df_p = \begin{bmatrix} -a\sin u \sin v & a\cos u \cos v \\ b\cos u \sin v & b\sin u \cos v \\ 0 & -c\sin v \end{bmatrix}$$

$f_u = \partial f/\partial u$ $\quad$ $f_v = \partial f/\partial v$

(b) Geometric meaning of $Df_p : T_p(\mathbb{R}^2) \to T_{f(p)}(\mathbb{R}^3)$

$$Df_p = \underbrace{\begin{bmatrix} \frac{\delta f}{\delta u} & \frac{\delta f}{\delta v} \end{bmatrix}}_{\text{Jacobian}} \in \mathbb{R}^{3\times 2}$$

→ Intuitively differential map Df_p of a parametrized surface tells us how tangent vectors in the domain gets mapped to tangent vectors on the surface

→ vectors $\frac{\delta f}{\delta u}$, $\frac{\delta f}{\delta v}$ lie on the tangent plane at point $p = (u, v)$ on the surface. They form a basis vector of the tangent plane at point $f(u, v)$.

→ As point $p \in \mathbb{R}^2$ moves in the domain space along direction $X = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$, the image point on surface fp will move along the direction $Df_p X$.

(d) Normal vector of tangent plane at p

$$N_p = N(u,v) = \frac{f_u \times f_v}{\| f_u \times f_v \|}$$ where $f_u = \delta f / \delta u$

$f_v = \delta f / \delta v$

$$f_u \times f_v = \begin{bmatrix} -a \sin u \sin v \\ b \cos u \sin v \\ 0 \end{bmatrix} \times \begin{bmatrix} a \cos u \cos v \\ b \sin u \cos v \\ -c \sin v \end{bmatrix}$$

$$= \begin{vmatrix} i & j & k \\ -a \sin u \sin v & b \cos u \sin v & 0 \\ a \cos u \cos v & b \sin u \cos v & -c \sin v \end{vmatrix}$$

$$= i(-bc \cos u \sin^2 v - 0) - j(ac \sin u \sin^2 v - 0)$$

$$+ k(-ab \sin^2 u \sin v \cos v - ab \cos^2 u \sin v \cos v)$$

$$= i\left(-bc\cos u\sin^2 v\right) - j\left(ac\sin u\sin 2v\right) - k\left(ab\sin v\cos v\right)$$

$$\|f_u \times f_v\| = \sqrt{b^2c^2\cos^2 u\sin^4 v + a^2c^2\sin^2 u\sin^4 v + a^2b^2\sin^2 v\cos^2 v}$$

when $a=1,\ b=1,\ c=\frac{1}{2}$

$$\|f_u \times f_v\| = \sqrt{\frac{1}{4}\sin^4 v + \sin^2 v\cos^2 v}$$

$$= |\sin v|\sqrt{\frac{1}{4}\sin^2 v + \cos^2 v}$$

$$N_p = N(u,v) = \frac{f_u \times f_v}{\|f_u \times f_v\|}$$

$$= \frac{i\left(-\frac{1}{2}\cos u\sin^2 v\right) - j\left(\frac{1}{2}\sin u\sin^2 v\right) - k\left(\sin v\cos v\right)}{|\sin v|\left(\frac{1}{4}\sin^2 v + \cos^2 v\right)^{1/2}}$$

(e) orthonormal basis vectors of the tangent space at $f(p)$ when $p = \left(\frac{\pi}{4}, \frac{\pi}{6}\right)$

$$Df_p = [f_u, f_v] = \left[\frac{\partial f}{\partial u}, \frac{\partial f}{\partial v}\right]_{3\times 2}$$

Now, $$f_u \cdot f_v = \begin{bmatrix} -a\sin u \sin v \\ b\cos u \sin v \\ 0 \end{bmatrix} \cdot \begin{bmatrix} a\cos u \cos v \\ b\sin u \cos v \\ -c\sin v \end{bmatrix}$$

$$= -a^2 \sin u \cos u \sin v \cos v + b^2 \sin u \cos u \sin v \cos v$$

$$= 0 \qquad [\because a = b = 1]$$

Thus $f_u \perp f_v$ and $\langle f_u, f_v \rangle = 0$

f_u, f_v are orthogonal basis vectors

$$\|f_u\| = \sqrt{a^2 \sin^2 u \sin^2 v + b^2 \cos^2 u \sin^2 v} = |\sin v|$$

$$\|f_v\| = \sqrt{a^2\cos^2 u\cos^2 v + b^2\sin^2 u\cos^2 v + c^2\sin^2 v}$$

$$= \sqrt{\cos^2 v + c^2\sin^2 v} = \sqrt{\tfrac{1}{4}\sin^2 v + \cos^2 v}$$

Thus $\hat{f}_u = \dfrac{f_u}{\|f_u\|}$ and $\hat{f}_v = \dfrac{f_v}{\|f_v\|}$ are orthonormal basis vectors.

At $p_2(u,v) = (\pi/4, \pi/6)$

$$\|f_u\| = 0.5 \qquad \|f_v\| = \frac{\sqrt{13}}{4}$$

$$f_u = \begin{bmatrix} -1/2\sqrt{2} \\ \dfrac{1}{2\sqrt{2}} \\ 0 \end{bmatrix} \qquad f_v = \begin{bmatrix} \sqrt{3}/2\sqrt{2} \\ \sqrt{3}/2\sqrt{2} \\ -1/4 \end{bmatrix}$$

$$\hat{f}_u = \frac{f_u}{\|f_u\|} = \begin{bmatrix} -\frac{1}{4\sqrt{2}} \\ \frac{1}{4\sqrt{2}} \\ 0 \end{bmatrix}$$

$$\hat{f}_v = \frac{f_v}{\|f_v\|} = \frac{4}{\sqrt{13}} \begin{bmatrix} \sqrt{3}/2\sqrt{2} \\ \sqrt{3}/2\sqrt{2} \\ -1/4 \end{bmatrix} = \frac{1}{\sqrt{13}} \begin{bmatrix} \sqrt{6} \\ \sqrt{6} \\ -1 \end{bmatrix}$$

④ $p = \left[\frac{\pi}{4}, \frac{\pi}{6}\right] \quad v = [1, 0]$

Let $g_v(t) = (f \circ \gamma)(t)$ denotes the curve passing through p at $t=0$

ⓐ Arc Length $s(t) = \int_0^t \| g_v'(t) \| \, dt$

$$g_v'(t) = (f \circ \gamma)'(t) = f'(\gamma(t))$$

$$s(t) = \int_0^t \| f'(\gamma(t)) \| \, dt$$

$$f'(\gamma(t)) = \Big[-av_1 \sin(\gamma_1(t)) \sin(\gamma_2(t)) + av_2 \cos\gamma_1(t) \cos\gamma_2(t),$$

$$bv_1 \cos(\gamma_1(t)) \sin(\gamma_2(t)) + bv_2 \sin(\gamma_1(t)) \cos(\gamma_2(t)),$$

$$- c v_2 \sin(\gamma_2(t)) \Big]$$

$$= \left[-\sin(\gamma_1(t)) \sin(\gamma_2(t)), \cos(\gamma_1(t)) \sin(\gamma_2(t)), 0 \right]$$

$$= \left[-\sin\left(t+\frac{\pi}{4}\right)\sin\left(\frac{\pi}{6}\right),\ \cos\left(t+\frac{\pi}{4}\right)\sin\left(\frac{\pi}{6}\right),\ 0 \right]$$

$$= \left[-\frac{1}{2}\sin\left(t+\frac{\pi}{4}\right),\ \frac{1}{2}\cos\left(t+\frac{\pi}{4}\right),\ 0 \right]$$

$$s(t) = \int_0^t \sqrt{\frac{1}{4}\sin^2\left(t+\frac{\pi}{4}\right) + \frac{1}{4}\cos^2\left(t+\frac{\pi}{4}\right) + 0}\ dt$$

$$= \frac{1}{2}\int_0^t dt = \frac{t}{2}$$

Arc length $\boxed{s(t) = \frac{t}{2}}$

(b)

$$f(r(t)) = \begin{bmatrix} a\cos(r_1(t))\sin(r_2(t)) \\ b\sin(r_1(t))\sin(r_2(t)) \\ c\cos(r_2(t)) \end{bmatrix}$$

$$f(r(t)) = \begin{bmatrix} \cos(t+\pi/4)\sin(\pi/6) \\ \sin(t+\pi/4)\sin(\pi/6) \\ \frac{1}{2}\cos(\pi/6) \end{bmatrix}$$

As $s = t/2$ then

$$h_v(s) = \begin{bmatrix} \frac{1}{2}\cos(2s+\pi/4) \\ \frac{1}{2}\sin(2s+\pi/4) \\ \sqrt{3}/4 \end{bmatrix}$$

Arc length parametrization

(c) Normal vector of the curve at point $h_v(s)$

(1) — $\frac{dT(s)}{ds} = k(s)\,N(s)$, $k(s) \in \mathbb{R}$

$T(s), N(s) \in \mathbb{R}^3$

(2) — $\frac{dN(s)}{ds} = -k(s)\,T(s)$

$$T(s) = \frac{d}{ds} h_v(s) = \begin{bmatrix} -\sin(2s + \pi/4) \\ \cos(2s + \pi/4) \\ 0 \end{bmatrix}$$

$$\frac{dT(s)}{ds} = \begin{bmatrix} -2\cos(2s + \pi/4) \\ -2\sin(2s + \pi/4) \\ 0 \end{bmatrix} = k(s) \underbrace{\begin{bmatrix} N_x(s) \\ N_y(s) \\ N_z(s) \end{bmatrix}}_{N(s)}$$

$N_z(s) = 0$ $\quad N_x(s) = \dfrac{-2\cos(2s + \pi/4)}{k(s)}$

$N_y(s) = \dfrac{-2\sin(2s + \pi/4)}{k(s)}$

$$\frac{dN(s)}{ds} = -k(s)\,T(s) = -k(s)\begin{bmatrix} -\sin(2s+\pi/4) \\ \cos(2s+\pi/4) \\ 0 \end{bmatrix}$$

$$\frac{dN_x(s)}{ds} = -2\left[-k(s)\sin(2s+\pi/4)\,2 - k'(s)\cos(2s+\pi/4)\right]$$

$$[k(s)]^2$$

$$\frac{d}{ds}N_x(s) = k(s)\sin(2s+\pi/4)$$

comparing

$$k'(s) = 0 \quad \forall s$$

$$\frac{4\sin(2s+\pi/4)}{k(s)} = k(s)\sin(2s+\pi/4)$$

$$\Rightarrow \boxed{k(s) = \pm 2}$$

Now, $$\frac{dN_y(s)}{ds} = -2\left[k(s)\cos(2s+\pi/4)\,2 - k'(s)\sin(2s+\pi/4)\right]$$

$$[k(s)]^2$$

$$\frac{dN_y(s)}{ds} = -k(s)\cos\left(2s + \pi/4\right)$$ ← comparing

$$k'(s) = 0 \qquad \frac{4}{k(s)} = k(s) \Rightarrow k(s) = \pm 2$$

For $k(s) = 2$

$$N(s) = \begin{bmatrix} -\cos\left(2s + \pi/4\right) \\ -\sin\left(2s + \pi/4\right) \\ 0 \end{bmatrix}$$

For $k(s) = -2$

$$N(s) = \begin{bmatrix} \cos\left(2s + \pi/4\right) \\ \sin\left(2s + \pi/4\right) \\ 0 \end{bmatrix}$$

(5) Normal at $p = N_p$

(a) Differential of normal $DN_p = \left[\frac{\delta N}{\delta u}, \frac{\delta N}{\delta v}\right]_{3\times 2}$

$$N_p = N(u,v) = \frac{i\left(-\frac{1}{2}\cos u \sin^2 v\right) - j\left(\frac{1}{2}\sin u \sin^2 v\right) - k\left(\sin v \cos v\right)}{|\sin v|\sqrt{\frac{1}{4}\sin^2 v + \cos^2 v}}$$

$$\frac{\delta N(u,v)}{\delta u} = \frac{1}{|\sin v|\left(\frac{1}{4}\sin^2 v + \cos^2 v\right)^{1/2}} \begin{bmatrix} \frac{1}{2}\sin u \sin^2 v \\ -\frac{1}{2}\cos u \sin^2 v \\ 0 \end{bmatrix}$$

(*) $\frac{\delta N(u,v)}{\delta v}$ is computed using wolframalpha.com

$$\frac{\delta N(u,v)}{\delta v}[0] = \frac{-\frac{4}{3}\cos u \sin v \cos v}{\left(\cos 2v + \frac{5}{3}\right)\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)^{1/2}}$$

$$\frac{\delta N(u,v)}{\delta v}[1] = \frac{-\frac{4}{3}\sin u \sin v \cos v}{\left(\cos 2v + \frac{5}{3}\right)\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)^{1/2}}$$

$$\frac{\delta N(u,v)}{\delta v}[2] = \frac{0.25 \sin^4 v}{\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)^{3/2}}$$

$$\frac{\delta N(u,v)}{\delta v} = \frac{1}{\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)^{1/2}} \begin{bmatrix} \frac{-4\cos u \sin v \cos 2v}{(3\cos 2v + 5)} \\ \frac{-4\sin u \sin v \cos v}{(3\cos 2v + 5)} \\ \frac{0.25 \sin^4 v}{\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v} \end{bmatrix}$$

$$\frac{\delta N(u,v)}{\delta v}[0] = \frac{\left\{\cos u \cot v \left[\sin v\left(0.125\sin^2 v + 0.5\cos^2 v\right)\right.\right.}{\left.\left.\frac{d}{dv}|\sin v| + |\sin v|\left(-0.625\sin^2 v - \cos^2 v\right)\right]\right\}}{\left(\frac{1}{4}\sin^2 v + \cos^2 v\right)^{3/2}}$$

$$\frac{\delta N(u,v)}{\delta v}[1] = \frac{\left[\sin u \cot v \left\{\sin v\left(0.125\sin^2 v + 0.5\cos^2 v\right)\right.\right.}{\left.\left.\frac{d}{dv}|\sin v| + |\sin v|\left(-0.625\sin^2 v - \cos^2 v\right)\right\}\right]}{\left(\frac{1}{4}\sin^2 v + \cos^2 v\right)^{3/2}}$$

$$\frac{\delta N(u,v)}{\delta v}[2] = \frac{(0.25 - \cot^4 v)|\sin v| + \cos v \cot v\left(\cot^2 v + 0.25\right)\frac{d}{dv}|\sin v|}{\left(\cot^4 v + 1/4\right)\left(\frac{1}{4}\sin^2 v + \cos^2 v\right)^{1/2}}$$

(b) Eigenvectors of shape operator at p

$\exists\ S \in \mathbb{R}^{2\times 2}$ and $S = \begin{bmatrix} s_1 & s_2 \\ s_3 & s_4 \end{bmatrix}$ s.t.

$$\boxed{DN_p = Df_p S}$$

$$Df_p S = \begin{bmatrix} -\sin u \sin v & \cos u \cos v \\ \cos u \sin v & \sin u \cos v \\ 0 & -\frac{1}{2}\sin v \end{bmatrix} \begin{bmatrix} s_1 & s_2 \\ s_3 & s_4 \end{bmatrix}$$

$$= \begin{bmatrix} -s_1 \sin u \sin v + s_3 \cos u \cos v & -s_2 \sin u \sin v + s_4 \cos u \cos v \\ s_1 \cos u \sin v + s_3 \sin u \cos v & s_2 \cos u \sin v + s_4 \sin u \cos v \\ -\frac{s_3}{2}\sin v & -\frac{s_4}{2}\sin v \end{bmatrix}$$

Comparing element-wise with DNp.

$$-\frac{s_3 \sin v}{2} = 0 \quad \Rightarrow \quad \boxed{s_3 = 0}$$

$$-s_1 \sin u \sin v + \underbrace{s_3 \cos u \cos v}_{0} = \frac{\sin u \sin^2 v}{2|\sin v|\sqrt{\frac{1}{4}\sin^2 v + \cos^2 v}}$$

$$\Rightarrow \quad \boxed{s_1 = \frac{-\sin v}{2|\sin v|\sqrt{\frac{1}{4}\sin^2 v + \cos^2 v}}}$$

$$-\frac{s_4}{2}\sin v = \frac{1}{4}\frac{\sin^4 v}{\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)^{3/2}}$$

$$\Rightarrow \quad \boxed{s_4 = \frac{-\sin^3 v}{2\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)^{3/2}}}$$

$$-s_2 \sin u \sin v + s_4 \cos u \cos v = \frac{-4 \cos u \cos v \sin v}{(3\cos 2v + 5)\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)^{1/2}}$$

Now, $S_4 \cos u \cos v = \dfrac{-\sin^3 v \cos u \cos v}{2\left(\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v\right)\sqrt{\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v}}$

$$= \frac{-2\sin v \cos v \cos u}{\left(4\cos^2 v + \sin^2 v\right)\sqrt{\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v}}$$

$$= \frac{-4\cos u \cos v \sin v}{\left(3\cos 2v + 5\right)\sqrt{\frac{1}{4}\sin^4 v + \sin^2 v \cos^2 v}}$$

Thus, $\boxed{S_2 = 0}$

Shape operator $S = \begin{bmatrix} S_1 & 0 \\ 0 & S_4 \end{bmatrix}_{2\times 2}$ ← diagonal matrix

$$Sv = \lambda v$$

$v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and corresponding $\lambda_1 = s_1$

$v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and corresponding $\lambda_2 = s_2$

→ eigenvectors of shape operator S.

(c) Principal curvature directions in the tangent plane at $p = \left(\frac{\pi}{4}, \frac{\pi}{6}\right)$

$$Df_p = \begin{bmatrix} f_u & f_v \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial u}, & \frac{\partial f}{\partial v} \end{bmatrix}_{3\times 2}$$

$$= \begin{bmatrix} -a\sin u \sin v & a\cos u \cos v \\ b\cos u \sin v & b\sin u \cos v \\ 0 & -c\sin v \end{bmatrix}$$

$$Df_p = \begin{bmatrix} -1/2\sqrt{2} & \sqrt{3}/2\sqrt{2} \\ 1/2\sqrt{2} & \sqrt{3}/2\sqrt{2} \\ 0 & -1/4 \end{bmatrix}$$

1st principal curvature direction $= Df_p v_1$

$$\begin{bmatrix} -1/2\sqrt{2} & \sqrt{3}/2\sqrt{2} \\ 1/2\sqrt{2} & \sqrt{3}/2\sqrt{2} \\ 0 & -1/4 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1/2\sqrt{2} \\ 1/2\sqrt{2} \\ 0 \end{bmatrix}$$

2nd principal curvature direction $= Df_p v_2$

$$\begin{bmatrix} -1/2\sqrt{2} & \sqrt{3}/2\sqrt{2} \\ 1/2\sqrt{2} & \sqrt{3}/2\sqrt{2} \\ 0 & -1/4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{3}/2\sqrt{2} \\ \sqrt{3}/2\sqrt{2} \\ -1/4 \end{bmatrix}$$

(d) $\langle Df_p v_1 , Df_p v_2 \rangle$ at $p = \left[\frac{\pi}{4}, \frac{\pi}{6}\right]$

$= 0$

Thus, the principal curvature directions are orthogonal.

→ the principal curvatures direction also form the basis vecton of the tangent plane at $\left[\frac{\pi}{4}, \frac{\pi}{6}\right]$

problem2

October 23, 2022

[3]:
```
#!pip3 install open3d
```

[3]:
```
from math import *
import open3d
import numpy as np
import matplotlib.pyplot as plt
```

[4]:
```
a, b, c = 1, 1, 0.5
```

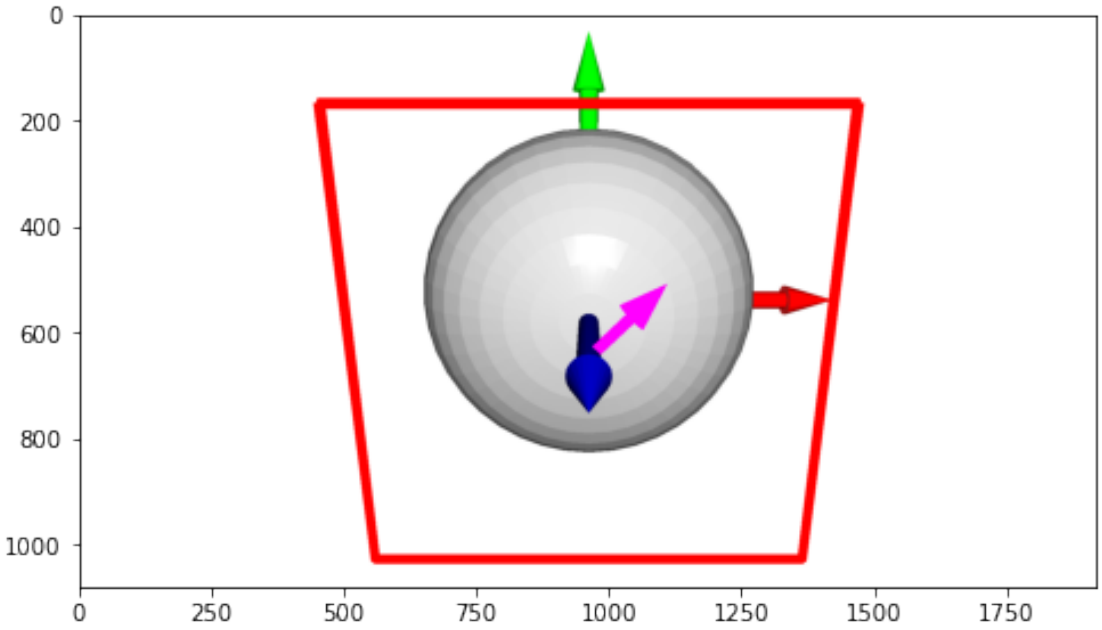
[74]:
```
# These are some convenient functions to create open3d geometries and plot them
# The viewing direction is fine-tuned for this problem, you should not change␣
 ↪them
vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)

def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    view_ctl.set_up((0, 1e-4, 1))
    view_ctl.set_front((0, 0.5, 2))
    view_ctl.set_lookat((0, 0, 0))
    # do not change this view point
    vis.update_renderer()
    img = vis.capture_screen_float_buffer(True)
    plt.figure(figsize=(8,6))
    plt.imshow(np.asarray(img)[::-1, ::-1])
    for g in geoms:
        vis.remove_geometry(g)

def create_arrow_from_vector(origin, vector):
    '''
    origin: origin of the arrow
    vector: direction of the arrow
    '''
    v = np.array(vector)
    v /= np.linalg.norm(v)
```

```
    z = np.array([0,0,1])
    angle = np.arccos(z@v)

    arrow = open3d.geometry.TriangleMesh.create_arrow(cylinder_radius=0.025,␣
 ↪cone_radius=0.07, cylinder_height=0.25,
                                                      cone_height=0.2)
    arrow.paint_uniform_color([1,0,1])
    T = np.eye(4)
    T[:3, 3] = np.array(origin)
    T[:3,:3] = open3d.geometry.get_rotation_matrix_from_axis_angle(np.cross(z,␣
 ↪v) * angle)
    arrow.transform(T)
    return arrow

def create_ellipsoid(a,b,c):
    sphere = open3d.geometry.TriangleMesh.create_sphere()
    sphere.transform(np.diag([a,b,c,1]))
    sphere.compute_vertex_normals()
    return sphere


def create_lines(points):
    lines = []
    for p1, p2 in zip(points[:-1], points[1:]):
        height = np.linalg.norm(p2-p1)
        center = (p1+p2) / 2
        d = p2-p1
        d /= np.linalg.norm(d)
        axis = np.cross(np.array([0,0,1]), d)
        axis /= np.linalg.norm(axis)
        angle = np.arccos(np.array([0,0,1]) @ d)
        R = open3d.geometry.get_rotation_matrix_from_axis_angle(axis * angle)

        T = np.eye(4)
        T[:3,:3]=R
        T[:3,3] = center
        cylinder = open3d.geometry.TriangleMesh.create_cylinder(0.02, height)
        cylinder.transform(T)
        cylinder.paint_uniform_color([1,0,0])
        lines.append(cylinder)
    return lines
```

[75]:
```
# exapmle code to draw ellipsoid, curve, and arrows
ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
```

```
arrow = create_arrow_from_vector([0.,0.,1.], [1.,1.,0.])
rectangular_curve = create_lines(np.array([[1,1,1], [-1,1,1], [-1,-1,1],␣
↪[1,-1,1], [1,1,1]], dtype=np.float64))
draw_geometries([ellipsoid, cf, arrow] + rectangular_curve)
```

```
WARNING - 2022-10-23 20:09:28,994 - image - Clipping input data to the valid
range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



[76]:
```
def compute_Fp(p):
    u, v = p
    fuv = [a * cos(u) * sin(v),
           b * sin(u) * sin(v),
           c * cos(v)
          ]

    return fuv
```

[77]:
```
def compute_GammaT(p, v, t):
    return v * t + p
```

[78]:
```
def compute_FofGammaT(p, v, t):
    gammaT = compute_GammaT(p, v, t)
    FofGammaT = [a * cos(gammaT[0]) * sin(gammaT[1]),
           b * sin(gammaT[0]) * sin(gammaT[1]),
           c * cos(gammaT[1])
           ]
```

```
    return FofGammaT
```

[79]:
```
def compute_FofGammaDashT(p, v, t):
    gammaT = compute_GammaT(p, v, t)

    FofGammaDashT = [-a * v[0] * sin(gammaT[0]) * sin(gammaT[1]) + a * v[1] *␣
 ↪cos(gammaT[0]) * cos(gammaT[1]),
                      b * v[0] * cos(gammaT[0]) * sin(gammaT[1]) + b * v[1] *␣
 ↪sin(gammaT[0]) * cos(gammaT[1]),
                      -c * v[1] * sin(gammaT[1])
                     ]
    return FofGammaDashT
```

0.0.1 2.1 Plot $f(\gamma(t))$ and $f^{'}(\gamma(0))$ on the surface of the ellipsoid in $\mathbb{R}^3$ for $p = [1, 1]$ and $v = [1, 1]$

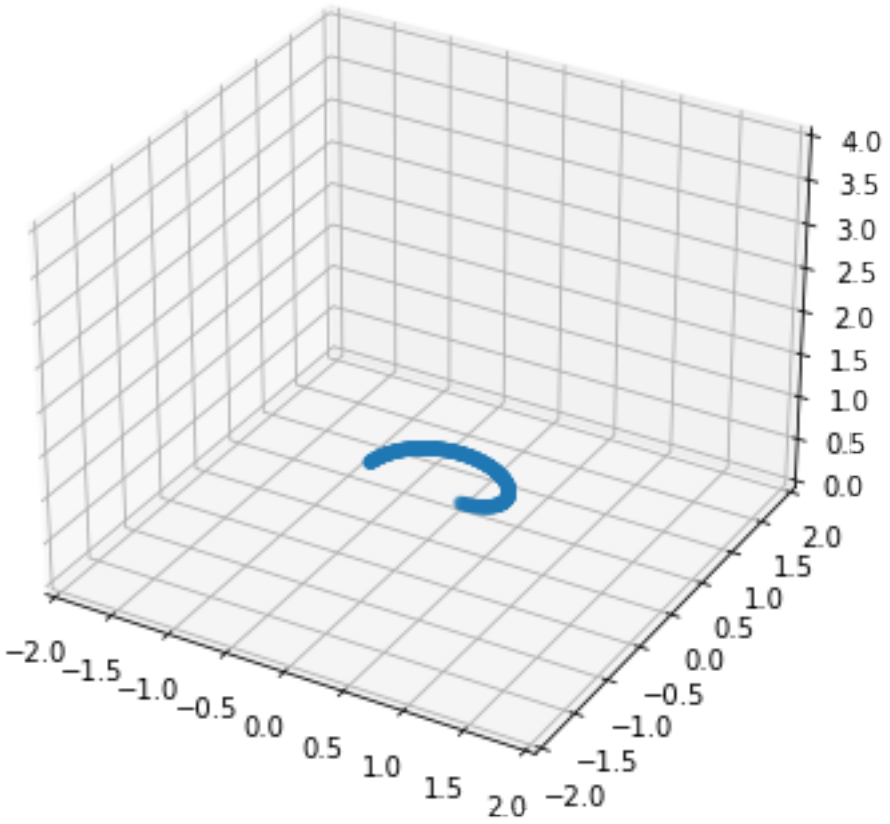
- Red curve represents $f(\gamma(t))$
- Pink/Magenta arrow represents $f^{'}(\gamma(0))$

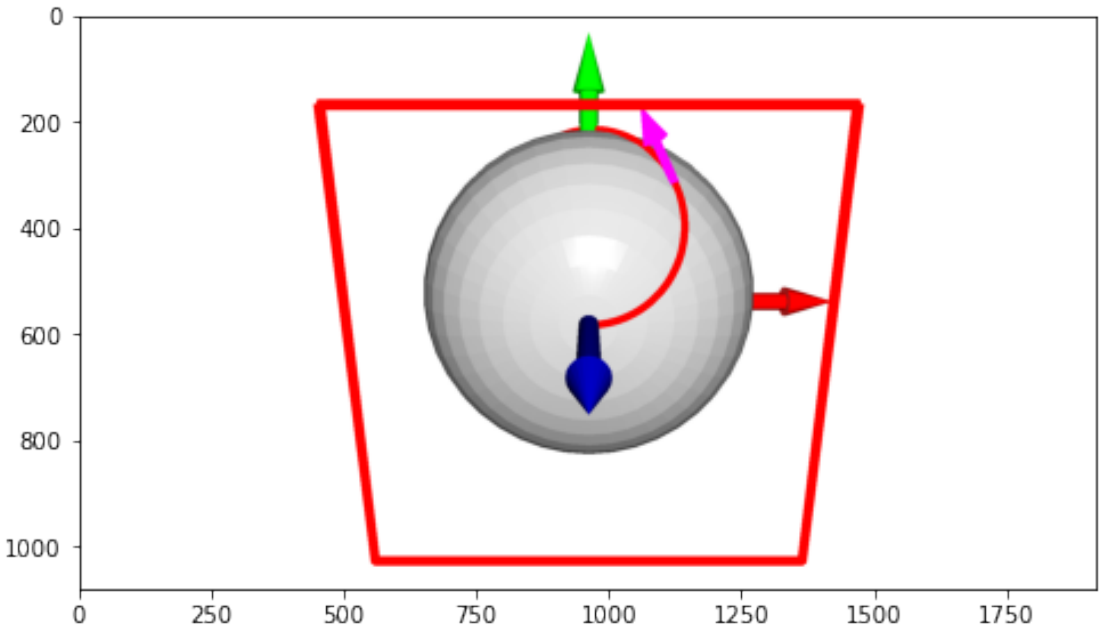
[80]:
```
def show_points(points):
    fig = plt.figure(figsize=(10,6))
    ax = fig.gca(projection='3d')
    ax.set_xlim3d([-2, 2])
    ax.set_ylim3d([-2, 2])
    ax.set_zlim3d([0, 4])
    ax.scatter(points[0], points[2], points[1])
```

[81]:
```
p = np.array([1.0, 1.0])
v = np.array([1.0, 1.0])

FofGammaT_pts = []

t_range = np.arange(-1, 1.00001, 0.01)
for t in t_range:
    FofGammaT = compute_FofGammaT(p, v, t)
    FofGammaT_pts.append(FofGammaT)

FofGammaT_pts = np.array(FofGammaT_pts)

show_points(FofGammaT_pts.T)

arrow = create_arrow_from_vector([cos(1)*sin(1), sin(1)*sin(1), 0.5*cos(1)],␣
 ↪[cos(2), sin(2), -0.5*sin(1)])
curve_FofGammaT = create_lines(FofGammaT_pts)
```

```
rectangle_curve = create_lines(np.array([[1,1,1], [-1,1,1], [-1,-1,1],␣
↪[1,-1,1], [1,1,1]], dtype=np.float64))
draw_geometries([ellipsoid, cf, arrow] + curve_FofGammaT + rectangle_curve)
```

```
WARNING - 2022-10-23 20:09:43,889 - image - Clipping input data to the valid
range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

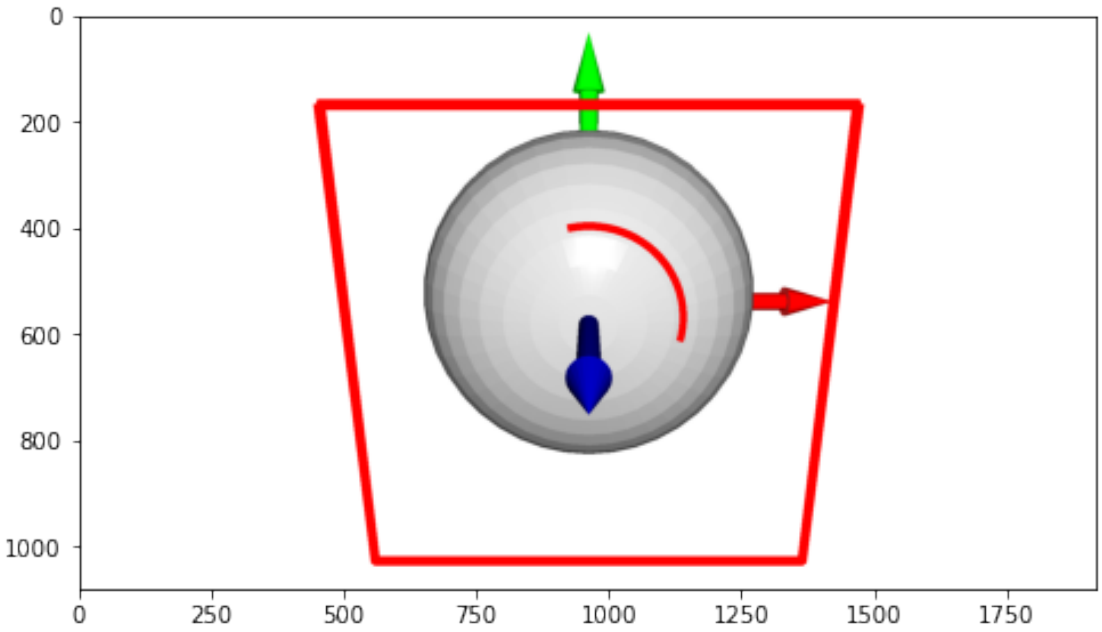




0.0.2 2.2 Let $p = [\pi/4, \pi/6]$ and $v = [1, 0]$. Draw the curve of $f(\gamma(t))$ on the surface of the ellipsoid.

[82]:
```
p = np.array([pi/4, pi/6])
v = np.array([1.0, 0])
t_range = np.arange(-1, 1.00001, 0.01)
# print(t_range)

# draw the curve of (f o \gamma)(t) on ellipsoid surface
curve_pts = []

for t in t_range:
    FofGammaT = compute_FofGammaT(p, v, t)
    curve_pts.append(FofGammaT)

curve_pts = np.array(curve_pts)
# print(curve_pts.shape)

rectangle_curve = create_lines(np.array([[1,1,1], [-1,1,1], [-1,-1,1],␣
↪[1,-1,1], [1,1,1]], dtype=np.float64))
curve_FofGammaT = create_lines(curve_pts)
draw_geometries([ellipsoid, cf] + curve_FofGammaT + rectangle_curve)
```

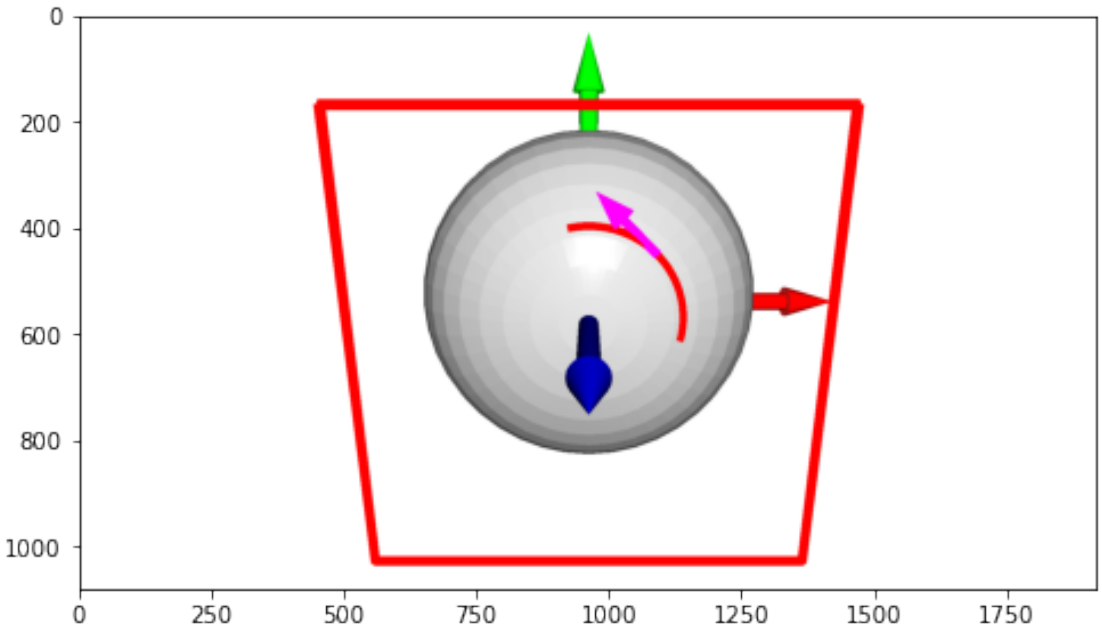
```
WARNING - 2022-10-23 20:09:48,512 - image - Clipping input data to the valid
range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



0.0.3 2.3 (c) Draw $Df_p(v)$ on the ellipsoid when $p = [\pi/4, \pi/6]$ and $v = [1, 0]$

[83]:
```
p = np.array([pi/4, pi/6])
v = np.array([1.0, 0])

fp = compute_Fp(p)
Dfpv = compute_FofGammaDashT(p, v, t=0)
# print(Dfpv)

rectangle_curve = create_lines(np.array([[1,1,1], [-1,1,1], [-1,-1,1],␣
↪[1,-1,1], [1,1,1]], dtype=np.float64))
arrow = create_arrow_from_vector(origin=fp, vector=Dfpv)
draw_geometries([ellipsoid, cf, arrow] + curve_FofGammaT + rectangle_curve)
```

```
WARNING - 2022-10-23 20:09:52,707 - image - Clipping input data to the valid
range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



0.0.4 2.3 (e) Give an orthonormal bases of the tangent space at $f(p)$ when $p = [\pi/4, \pi/6]$. Draw the orthonormal bases on the ellipsoid

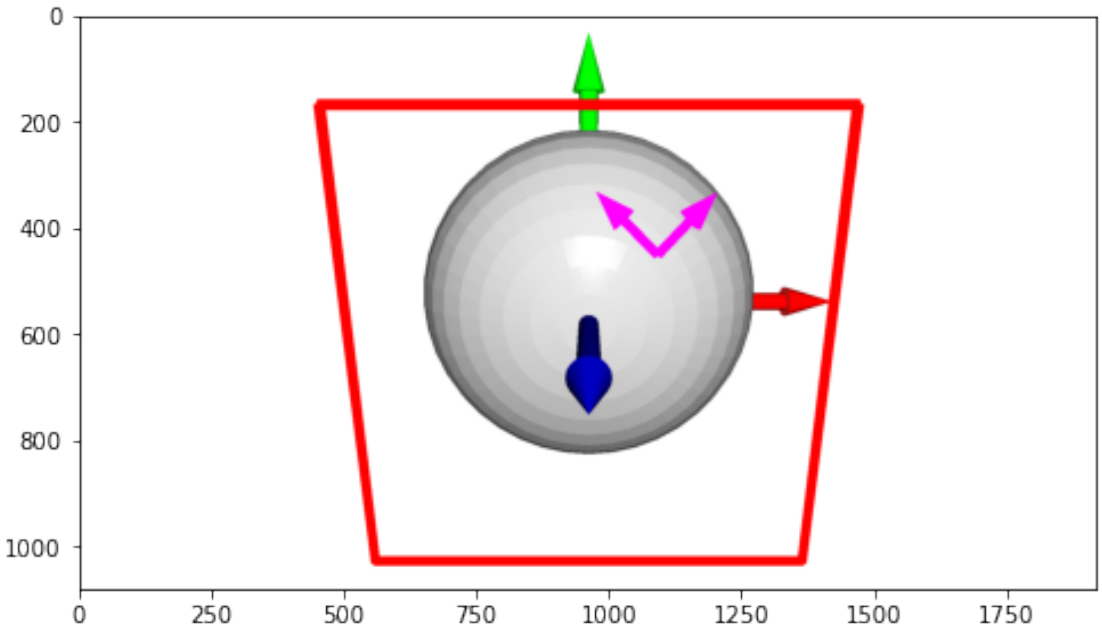
[84]:
```
def compute_dfdu_dfdv(p):
    u, v = p
    dfdu = [-a * sin(u) * sin(v), b * cos(u) * sin(v), 0]
    dfdv = [a * cos(u) * cos(v), b * sin(u) * cos(v), -c * sin(v)]

    return dfdu, dfdv
```

[85]:
```
p = np.array([pi/4, pi/6])

fp = compute_Fp(p)
dfdu, dfdv = compute_dfdu_dfdv(p)

dfdu_mag = np.linalg.norm(dfdu)
dfdv_mag = np.linalg.norm(dfdv)

dfdu_hat = dfdu / dfdu_mag
dfdv_hat = dfdv / dfdv_mag

print("Orthonormal basis vectors: \n", "v1:", dfdu_hat, "\n", "v2:", dfdv_hat,␣
 ↪"\n")
```

```
print("Magnitude |v1|=", np.linalg.norm(dfdu_hat), "Magnitude |v2|=", np.linalg.
 ↪norm(dfdv_hat))
print("Dot product (v1.v2) =", np.dot(dfdu_hat, dfdv_hat))

rectangle_curve = create_lines(np.array([[1,1,1], [-1,1,1], [-1,-1,1],␣
 ↪[1,-1,1], [1,1,1]], dtype=np.float64))
arrow1 = create_arrow_from_vector(origin=fp, vector=dfdu_hat)
arrow2 = create_arrow_from_vector(origin=fp, vector=dfdv_hat)
draw_geometries([ellipsoid, cf, arrow1, arrow2] + rectangle_curve)
```

```
WARNING - 2022-10-23 20:12:10,334 - image - Clipping input data to the valid
range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Orthonormal basis vectors:
 v1: [-0.70710678  0.70710678  0.        ]
 v2: [ 0.67936622  0.67936622 -0.2773501 ]

Magnitude |v1|= 1.0 Magnitude |v2|= 1.0
Dot product (v1.v2) = 0.0
```

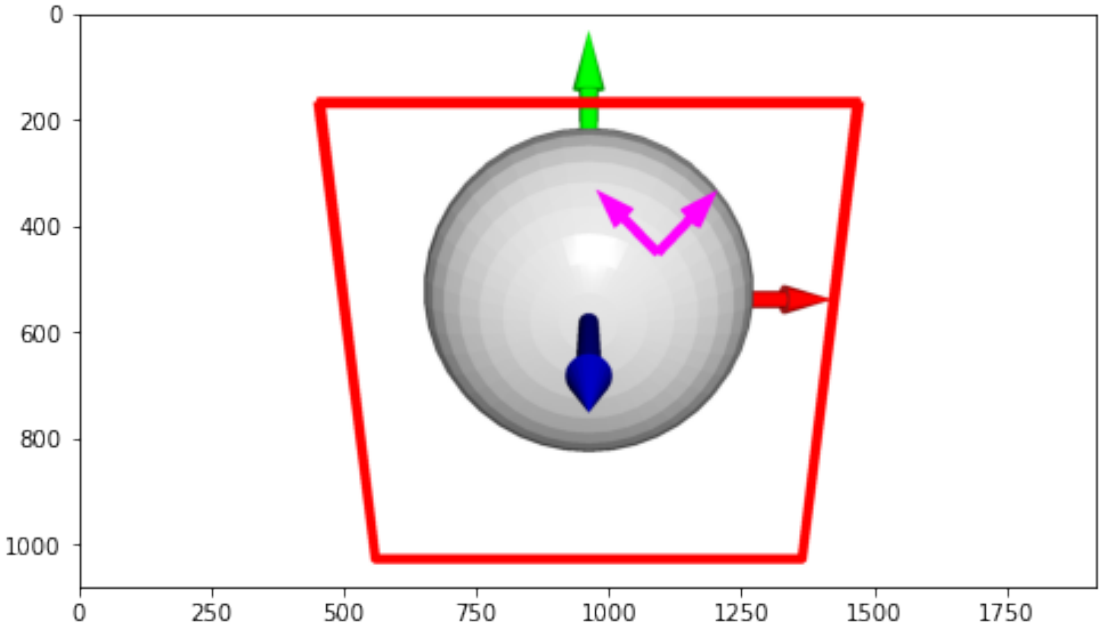


0.0.5 2.5 (c) Draw the two principal curvature directions in the tangent plane of the ellipsoid $p = [\frac{\pi}{4}, \frac{\pi}{6}]$

[86]:
```
# eigenvector of the shape operator
eigenvec1 = np.array([1.0, 0])
eigenvec2 = np.array([0, 1.0])

p = np.array([pi/4, pi/6])
fp = compute_Fp(p)

dfdu, dfdv = compute_dfdu_dfdv(p)
Dfp_mat = np.array([dfdu, dfdv]).T   # (3,2)

prin_curv_dir1 = np.matmul(Dfp_mat, eigenvec1)
prin_curv_dir1 = prin_curv_dir1 / np.linalg.norm(prin_curv_dir1)
prin_curv_dir2 = np.matmul(Dfp_mat, eigenvec2)
prin_curv_dir2 = prin_curv_dir2 / np.linalg.norm(prin_curv_dir2)

print("Dot product between two principal curvature directions:", np.
 ↪dot(prin_curv_dir1, prin_curv_dir2))

rectangle_curve = create_lines(np.array([[1,1,1], [-1,1,1], [-1,-1,1],␣
 ↪[1,-1,1], [1,1,1]], dtype=np.float64))
arrow1 = create_arrow_from_vector(origin=fp, vector=prin_curv_dir1)
arrow2 = create_arrow_from_vector(origin=fp, vector=prin_curv_dir2)
draw_geometries([ellipsoid, cf, arrow1, arrow2] + rectangle_curve)
```

```
WARNING - 2022-10-23 20:12:13,089 - image - Clipping input data to the valid
range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Dot product between two principal curvature directions: 0.0
```


0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

Part II : Mesh Processing

Problem 3: Taubin Matrix M_p for point p

$$M_p = \frac{1}{2\pi} \int_{-\pi}^{\pi} k_p(t_\theta)\, t_\theta\, t_\theta^T \, d\theta$$

Using hints, suppose T_1 & T_2 are principal curvature directions, $t_\theta = \cos\theta\, T_1 + \sin\theta\, T_2$

$$k_p(t_\theta) = k_p^1 \cos^2\theta + k_p^2 \sin^2\theta$$

k_p^1, k_p^2 → principal curvatures

Let $\cos\theta = c_\theta$ and $\sin\theta = c_\theta$

$T_1, T_2 \in \mathbb{R}^3$

$$t_\theta t_\theta^T = (c_\theta T_1 + s_\theta T_2)(c_\theta T_1 + s_\theta T_2)^T$$

$$= c_\theta^2 T_1 T_1^T + c_\theta s_\theta T_1 T_2^T + c_\theta s_\theta T_2 T_1^T + s_\theta^2 T_2 T_2^T$$

$$M_p = \frac{1}{2\pi} \int_{-\pi}^{\pi} k_p(t_\theta) \left[c_\theta^2 T_1 T_1^T + c_\theta s_\theta T_1 T_2^T + c_\theta s_\theta T_2 T_1^T + s_\theta^2 T_2 T_2^T \right] d\theta$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} (k_p' c_\theta^2 + k_p^2 s_\theta^2) \left[c_\theta^2 T_1 T_1^T + c_\theta s_\theta T_1 T_2^T + c_\theta s_\theta T_2 T_1^T + s_\theta^2 T_2 T_2^T \right] d\theta$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} \Big[c_\theta^4 k_p' T_1 T_1^T + \underbrace{c_\theta^3 s_\theta k_p' T_1 T_2^T}_{0} + \underbrace{c_\theta^3 s_\theta k_p' T_2 T_1^T}_{0} + c_\theta^2 s_\theta^2 k_p' T_2 T_2^T + s_\theta^2 c_\theta^2 k_p^2 T_1 T_1^T + \underbrace{s_\theta^3 c_\theta k_p^2 T_1 T_2^T}_{0} + \underbrace{s_\theta^3 c_\theta k_p^2 T_2 T_1^T}_{0}$$

$$+ S\theta^4 k_p^2 T_2 T_2^T \Big] d\theta$$

→ red underlined integrations will be 0.

$$M_p = \frac{1}{\pi} \int_0^{\pi} \Big[C\theta^4 k_p' T_1 T_1^T + C\theta^2 S\theta^2 k_p' T_2 T_2^T +$$

$$C\theta^2 S\theta^2 k_p^2 T_1 T_1^T + S\theta^4 k_p^2 T_2 T_2^T \Big] d\theta$$

$$= \frac{1}{\pi} \left[\frac{3\pi}{8} k_p' T_1 T_1^T + \frac{3\pi}{8} k_p^2 T_2 T_2^T + \right.$$

$$\left. \frac{\pi}{8} k_p' T_2 T_2^T + \frac{\pi}{8} k_p^2 T_1 T_1^T \right]$$

$$= T_1 T_1^T \left[\frac{3}{8} k_p' + \frac{1}{8} k_p^2 \right] +$$

$$T_2 T_2^T \left[\frac{3}{8} k_p^2 + \frac{1}{8} k_p' \right]$$

Let $\gamma_1 = \frac{3}{8} k_p' + \frac{1}{8} k_p^2$ & $\gamma_2 = \frac{3}{8} k_p^2 + \frac{1}{8} k_p'$

then $\boxed{M_p = \gamma_1 T_1 T_1^T + \gamma_2 T_2 T_2^T}$

Now, $\langle T_1, T_1 \rangle = 1$ and $\langle T_1, T_2 \rangle = 0$

$\langle T_2, T_2 \rangle = 1$ and $\|T_1\| = \|T_2\| = 1$

Since T_1 & T_2 are principal curvature directions

hence they are orthonormal to each other.

$$M_p T_1 = \gamma_1 T_1 \underbrace{T_1^T T_1}_{=1} + \gamma_2 T_2 \underbrace{T_2^T T_1}_{=0}$$

$$\boxed{M_p T_1 = \gamma_1 T_1}$$

$$M_p T_2 = \gamma_1 T_1 \underbrace{T_1^T T_2}_{=0} + \gamma_2 T_2 \underbrace{T_2^T T_2}_{=1}$$

$$\boxed{M_p T_2 = \gamma_2 T_2}$$

Thus, T_1 & T_2 are eigenvectors of M_p with eigenvalues γ_1 & γ_2 resp.

Proved

Surface normal at $p = \dfrac{T_1 \times T_2}{\|T_1 \times T_2\|}$

$$M_p \frac{(T_1 \times T_2)}{\|T_1 \times T_2\|} = \gamma_1 T_1 T_1^T \frac{(T_1 \times T_2)}{\|T_1 \times T_2\|} + \gamma_2 T_2 T_2^T \frac{(T_1 \times T_2)}{\|T_1 \times T_2\|}$$

$T_1^T (T_1 \times T_2) = 0$ & $T_2^T (T_1 \times T_2) = 0$

Thus,

$$M_p \frac{(T_1 \times T_2)}{\|T_1 \times T_2\|} = 0 \cdot \frac{(T_1 \times T_2)}{\|T_1 \times T_2\|}$$

Thus surface normal at p is an eigenvector of

M_p with corresponding eigenvalue $= 0$.

problem3

October 23, 2022

1 Part III: Mesh Processing

[1]:
```
import time
from tqdm.notebook import tqdm

import numpy as np
import matplotlib.pyplot as plt

import open3d
import trimesh
print(trimesh.__version__)
```

```
WARNING - 2022-10-23 21:52:27,920 - graph - graph-tool unavailable, some
operations will be much slower
WARNING - 2022-10-23 21:52:27,993 - assimp - pyassimp unavailable, using only
native loaders
WARNING - 2022-10-23 21:52:28,020 - creation - shapely.geometry.Polygon not
installed, some functions will not work!
```

```
2.3.13
```

[2]:
```
vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)

# Make sure you call this function to draw the points for proper viewing␣
↪direction
def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    view_ctl.set_up((0, 1, 0))
    view_ctl.set_front((0, 2, 1))
    view_ctl.set_lookat((0, 0, 0))
    view_ctl.set_zoom(1)
    # do not change this view point
    vis.update_renderer()
    img = vis.capture_screen_float_buffer(True)
```

```
    plt.figure(figsize=(8,6))
    plt.imshow(np.asarray(img))
    for g in geoms:
        vis.remove_geometry(g)

def create_arrow_from_vector(origin, vector):
    '''
    origin: origin of the arrow
    vector: direction of the arrow
    '''
    v = np.array(vector)
    v /= np.linalg.norm(v)
    z = np.array([0,0,1])
    angle = np.arccos(z@v)

    arrow = open3d.geometry.TriangleMesh.create_arrow(cylinder_radius=0.025,␣
↪cone_radius=0.05, cylinder_height=0.25,
                                                      cone_height=0.2)
    arrow.paint_uniform_color([1,0,1])
    T = np.eye(4)
    T[:3, 3] = np.array(origin)
    T[:3,:3] = open3d.geometry.get_rotation_matrix_from_axis_angle(np.cross(z,␣
↪v) * angle)
    arrow.transform(T)
    return arrow
```

[3]:
```
def compute_face_normals(mesh):
    num_faces = mesh.faces.shape[0]
    num_vertices = mesh.vertices.shape[0]

    face_normals = []

    for itr in tqdm(range(num_faces)):
        vertices_idx = mesh.faces[itr]   # (3,)
        vertices = mesh.vertices[vertices_idx]   # 3x3

        # compute edge vectors for each face
        edges = np.zeros((3,3))
        edges[0] = vertices[2] - vertices[1]
        edges[1] = vertices[0] - vertices[2]
        edges[2] = vertices[1] - vertices[0]

        # compute face orthonormal basis vectors using Gram-Schmidt␣
↪orthogonalization
        Dfp = np.zeros((3,2))
        Dfp[:,0] = edges[0]
```

```
#         assert(np.dot(edges[0], edges[0]) - np.linalg.norm(edges[0])**2 <=␣
 ↪1e-10)
        proj_component = (np.dot(edges[1], edges[0]) / np.dot(edges[0],␣
 ↪edges[0])) * edges[0]
        Dfp[:,1] = edges[1] - proj_component
        Dfp = Dfp / np.linalg.norm(Dfp, axis=0)
#         assert(np.dot(Dfp[:,0], Dfp[:,1]) <= 1e-10)

        # compute face normal
        face_normal = np.cross(Dfp[:,0], Dfp[:,1])
        face_normals.append(face_normal)

    face_normals = np.array(face_normals)
    return face_normals
```

1.1 Function to compute shape operator

[4]:
```
def compute_shape_operator(mesh):
    num_faces = mesh.faces.shape[0]
    num_vertices = mesh.vertices.shape[0]

    start_time = time.time()

    # compute face normals
#     face_normals = compute_face_normals(mesh)
    face_normals = mesh.face_normals

    # compute vertex normals
    vertex_normals = trimesh.geometry.mean_vertex_normals(num_vertices, mesh.
 ↪faces, face_normals)
#     pcd.estimate_normals(open3d.geometry.KDTreeSearchParamKNN(knn=50))

    principal_curv = np.zeros((num_faces, 2))

    for itr in tqdm(range(num_faces)):
        vertices_idx = mesh.faces[itr]   # (3,)
        vertices = mesh.vertices[vertices_idx]   # 3x3

        # compute edge vectors for each face
        edges = np.zeros((3,3))
        edges[0] = vertices[2] - vertices[1]
        edges[1] = vertices[0] - vertices[2]
        edges[2] = vertices[1] - vertices[0]

        # compute face orthonormal basis vectors using Gram-Schmidt
```

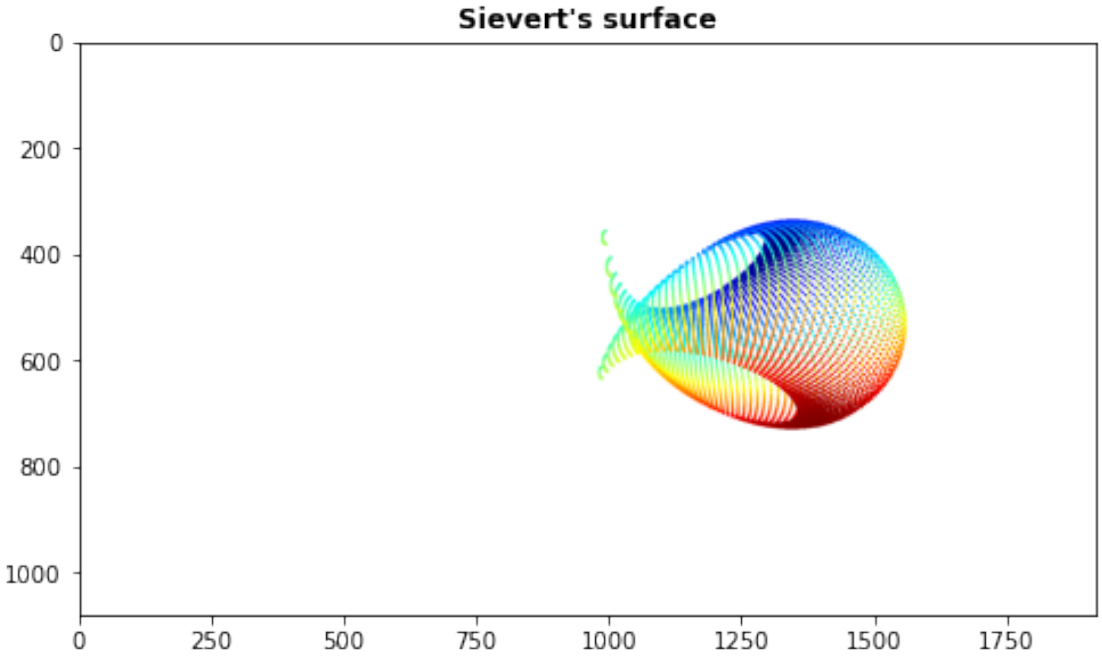
```
        Dfp = np.zeros((3,2))
        Dfp[:,0] = edges[0]
        proj_component = (np.dot(edges[1], edges[0]) / np.dot(edges[0],␣
↪edges[0])) * edges[0]
        Dfp[:,1] = edges[1] - proj_component
        Dfp = Dfp / np.linalg.norm(Dfp, axis=0)
#         assert(np.dot(Dfp[:,0], Dfp[:,1]) <= 1e-10)

        lhs1 = np.matmul(Dfp.T, edges[0])
        rhs1 = np.matmul(Dfp.T, vertex_normals[vertices_idx[2]] -␣
↪vertex_normals[vertices_idx[1]])

        lhs2 = np.matmul(Dfp.T, edges[1])
        rhs2 = np.matmul(Dfp.T, vertex_normals[vertices_idx[0]] -␣
↪vertex_normals[vertices_idx[2]])

        lhs3 = np.matmul(Dfp.T, edges[2])
        rhs3 = np.matmul(Dfp.T, vertex_normals[vertices_idx[1]] -␣
↪vertex_normals[vertices_idx[0]])

        A = np.array([lhs1, lhs2, lhs3]).T
        B = np.array([rhs1, rhs2, rhs3]).T

        S = np.matmul(np.matmul(B, A.T), np.linalg.inv(np.matmul(A, A.T)))

        principal_curv[itr], _ = np.linalg.eig(S)

    print("Time taken:", round(time.time() - start_time, 3), 's')

    return principal_curv
```

1.2 Load Sievert Object

[5]:
```
mesh_sievert = trimesh.load('../data/sievert.obj')
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(mesh_sievert.vertices)
# vertex_normals = pcd.estimate_normals(open3d.geometry.
↪KDTreeSearchParamKNN(knn=50))
# print("Vertex normals:", vertex_normals)
draw_geometries([pcd])
plt.title("Sievert's surface", fontweight ="bold")

print(mesh_sievert)
print("Vertices:", mesh_sievert.vertices.shape, "Faces:", mesh_sievert.faces.
↪shape,
```

```
      mesh_sievert.faces.min(), mesh_sievert.faces.max())
```

```
<trimesh.base.Trimesh object at 0x7f79a773e7c0>
Vertices: (10201, 3) Faces: (20000, 3) 0 10200
```



1.3 Compute principal, gaussian and mean curvatures for Sievert surface

[6]:
```
principal_curv_sievert = compute_shape_operator(mesh_sievert)
gauss_curv_sievert = np.multiply(principal_curv_sievert[:,0],␣
 ↪principal_curv_sievert[:,1])
mean_curv_sievert = 0.5 * np.add(principal_curv_sievert[:,0],␣
 ↪principal_curv_sievert[:,1])
```

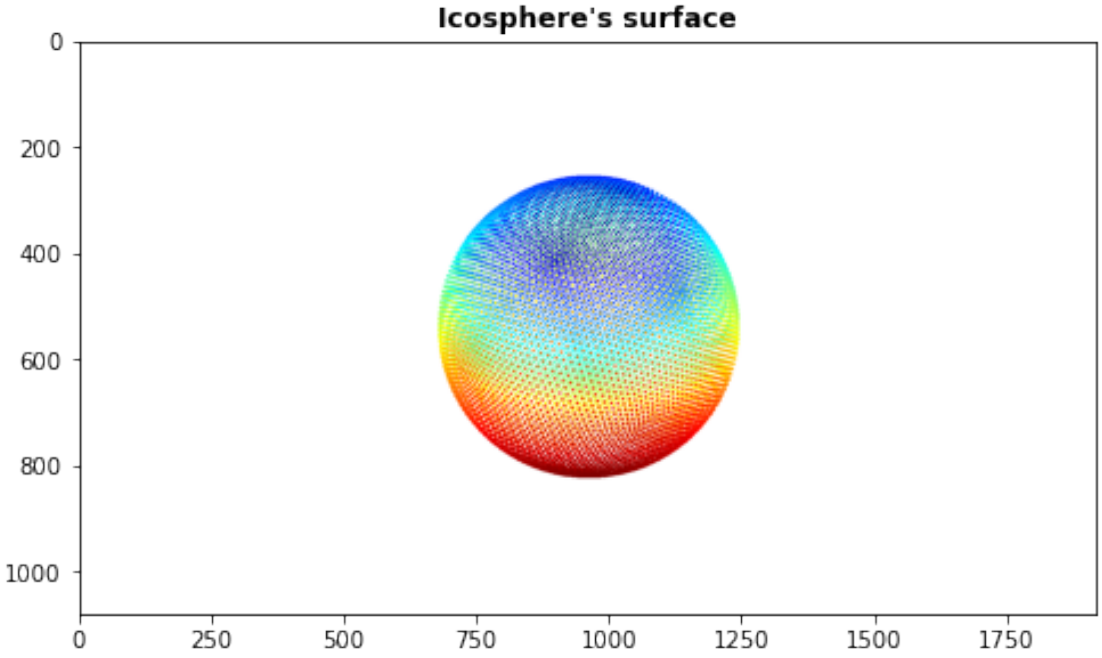
```
  0%|          | 0/20000 [00:00<?, ?it/s]
Time taken: 16.816 s
```

1.4 Load Icosphere Object

[8]:
```
mesh_icosphere = trimesh.load('../data/icosphere.obj')
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(mesh_icosphere.vertices)
draw_geometries([pcd])
```

```
plt.title("Icosphere's surface", fontweight ="bold")

print(mesh_icosphere)
print(mesh_icosphere.vertices.shape, mesh_icosphere.faces.shape, mesh_icosphere.
 ↪faces.min(), mesh_icosphere.faces.max())
```

```
<trimesh.base.Trimesh object at 0x7f79a62b4190>
(10242, 3) (20480, 3) 0 10241
```



1.5 Compute principal, gaussian and mean curvatures for Icosphere object

[9]:
```
principal_curv_icosphere = compute_shape_operator(mesh_icosphere)
gauss_curv_icosphere = np.multiply(principal_curv_icosphere[:,0],␣
 ↪principal_curv_icosphere[:,1])
mean_curv_icosphere = 0.5 * np.add(principal_curv_icosphere[:,0],␣
 ↪principal_curv_icosphere[:,1])
```

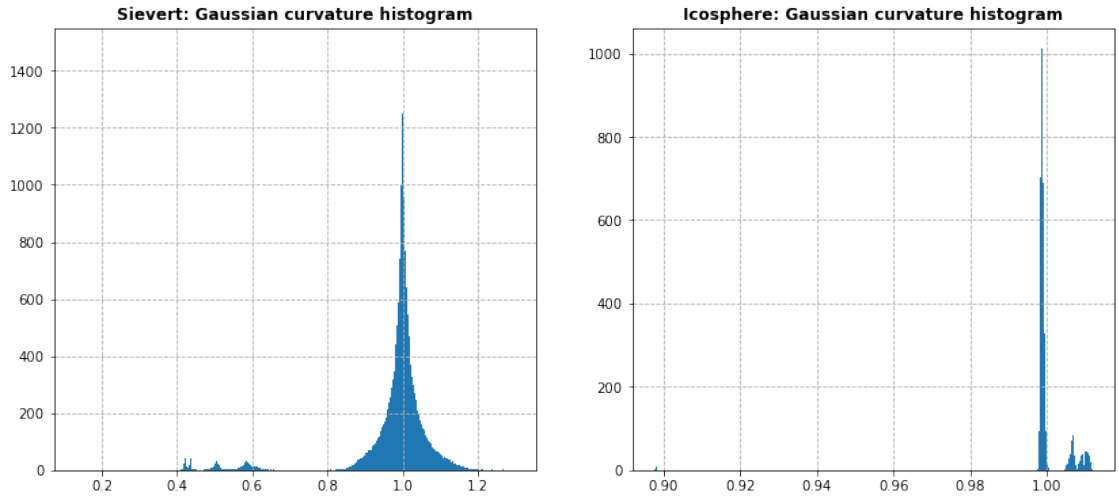
```
  0%|          | 0/20480 [00:00<?, ?it/s]
```

```
<ipython-input-4-7ce98e4b001a>:50: ComplexWarning: Casting complex values to
real discards the imaginary part
  principal_curv[itr], _ = np.linalg.eig(S)
```

```
Time taken: 15.44 s
```

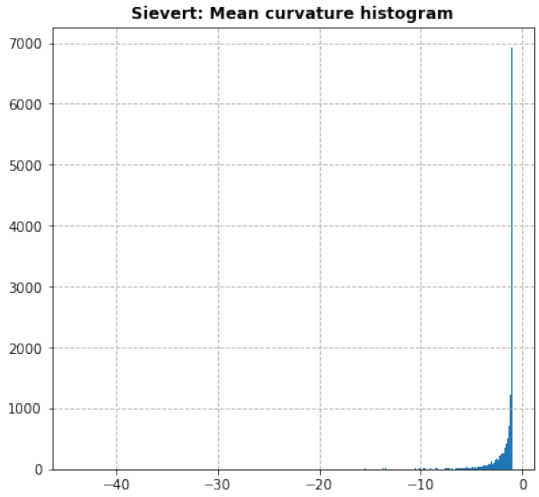
1.6 Plot gaussian curvature for Sievert & Icosphere object

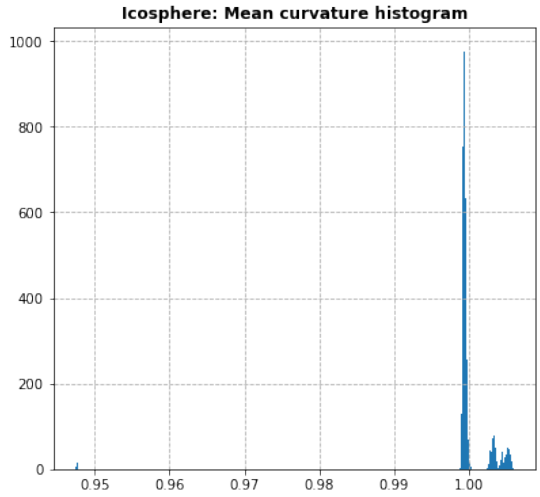
[10]:
```
fig = plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
plt.hist(gauss_curv_sievert, bins='auto')
plt.title("Sievert: Gaussian curvature histogram", fontweight ="bold")
plt.grid(linestyle='--')

plt.subplot(1,2,2)
plt.hist(gauss_curv_icosphere, bins='auto')
plt.title("Icosphere: Gaussian curvature histogram", fontweight ="bold")
plt.grid(linestyle='--')
```



1.7 Plot mean curvature for Sievert and Icosphere object

[11]:
```
fig = plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
_ = plt.hist(mean_curv_sievert, bins='auto')
plt.title("Sievert: Mean curvature histogram", fontweight ="bold")
plt.grid(linestyle='--')

plt.subplot(1,2,2)
_ = plt.hist(mean_curv_icosphere, bins='auto')
plt.title("Icosphere: Mean curvature histogram", fontweight ="bold")
plt.grid(linestyle='--')
```





1.7.1 - Comparing the gaussian curvatures for Sievert and Icosphere object, we can say that the sievert surface is locally isometric to some region on the icosphere object.

1.7.2 - The right semi-spherical part of sievert surface is locally isometric to the right part of icosphere

problem4

October 23, 2022

1 Part IV: Point Cloud Processing

[2]:
```
import time
import progressbar
from tqdm.notebook import tqdm

import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

import open3d
import trimesh
print("Trimesh version:", trimesh.__version__)
```

```
WARNING - 2022-10-23 21:55:52,603 - graph - graph-tool unavailable, some
operations will be much slower
WARNING - 2022-10-23 21:55:52,679 - assimp - pyassimp unavailable, using only
native loaders
WARNING - 2022-10-23 21:55:52,704 - creation - shapely.geometry.Polygon not
installed, some functions will not work!
```

```
Trimesh version: 2.3.13
```

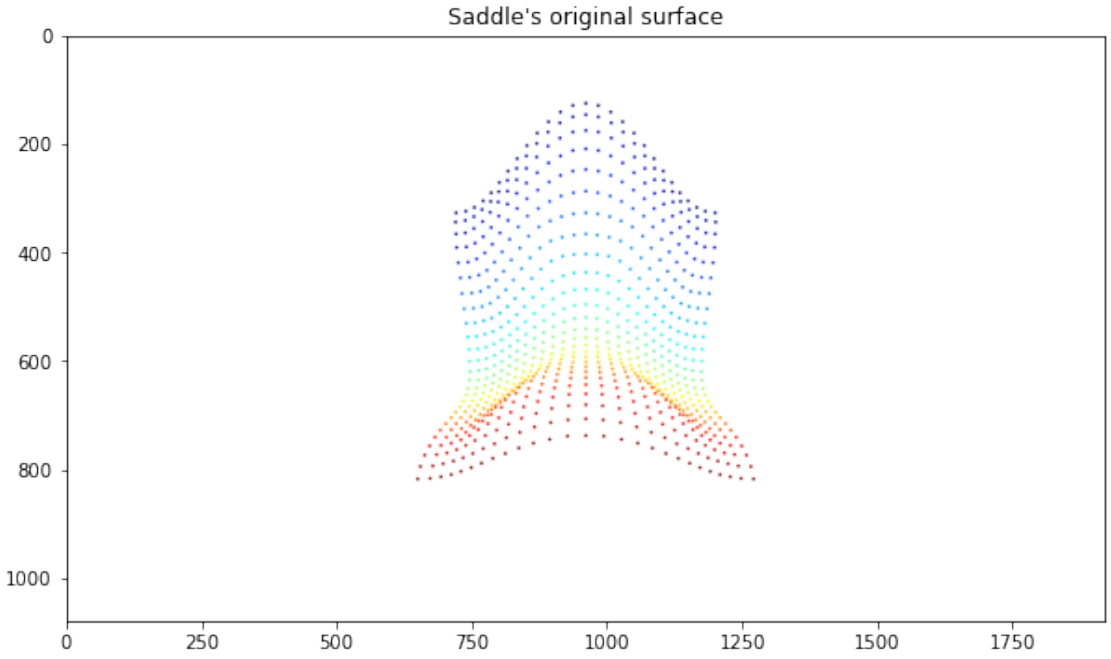
[12]:
```
vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)

# Make sure you call this function to draw the points for proper viewing␣
 ↪direction
def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    view_ctl.set_up((0, 1, 0))
    view_ctl.set_front((0, 2, 1))
    view_ctl.set_lookat((0, 0, 0))
    view_ctl.set_zoom(1)
    # do not change this view point
```

```
    vis.update_renderer()
    img = vis.capture_screen_float_buffer(True)
    plt.figure(figsize=(10,8))
    plt.imshow(np.asarray(img))
    for g in geoms:
        vis.remove_geometry(g)

def create_arrow_from_vector(origin, vector):
    '''
    origin: origin of the arrow
    vector: direction of the arrow
    '''
    v = np.array(vector)
    v /= np.linalg.norm(v)
    z = np.array([0,0,1])
    angle = np.arccos(z@v)

    arrow = open3d.geometry.TriangleMesh.create_arrow(cylinder_radius=0.02,␣
 ↪cone_radius=0.05, cylinder_height=0.12,
                                                      cone_height=0.1)
    arrow.paint_uniform_color([1,0,1])
    T = np.eye(4)
    T[:3, 3] = np.array(origin)
    T[:3,:3] = open3d.geometry.get_rotation_matrix_from_axis_angle(np.cross(z,␣
 ↪v) * angle)
    arrow.transform(T)
    return arrow
```

1.1 Load and display original saddle object

[13]:
```
mesh_saddle = trimesh.load('../data/saddle.obj')
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(mesh_saddle.vertices)
draw_geometries([pcd])
plt.title("Saddle's original surface")

print(mesh_saddle)
print(mesh_saddle.vertices.shape, mesh_saddle.faces.shape, mesh_saddle.faces.
 ↪min(), mesh_saddle.faces.max())
# print(mesh_saddle.vertices)
# print(mesh_saddle.faces)
```

```
<trimesh.base.Trimesh object at 0x7fd6c2be4880>
(729, 3) (676, 3) 0 728
```



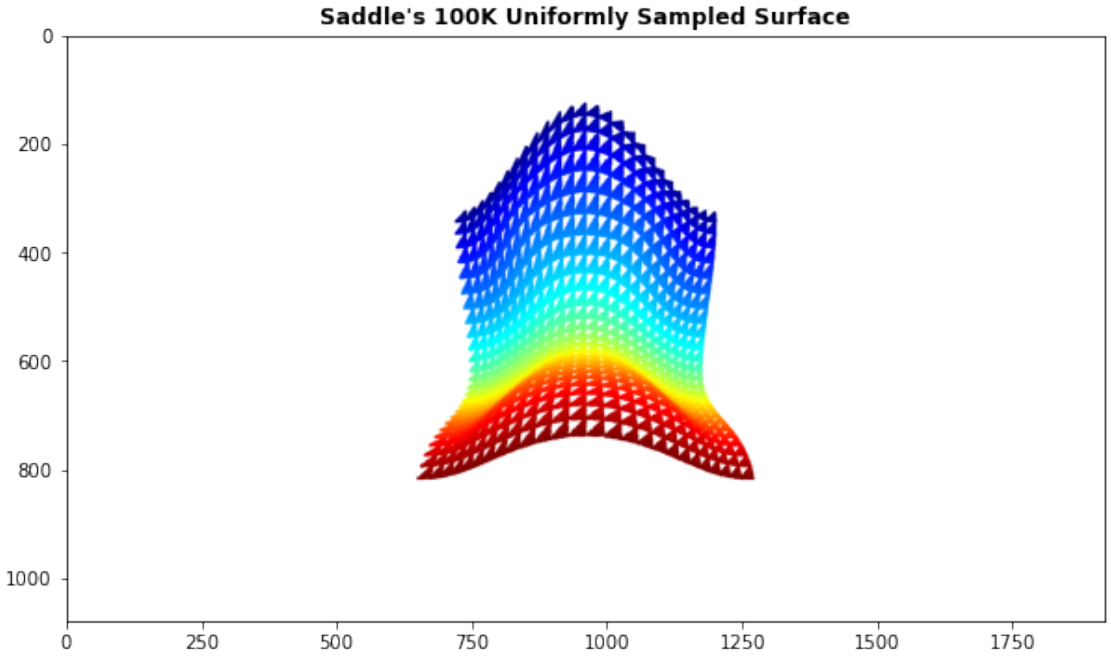
1.2 4.1 Sample 100k points uniformly on the surface

[36]:
```
# unif_samples = trimesh.sample.sample_surface(mesh_saddle,␣
 ↪count=num_unif_samples)
unif_samples = trimesh.sample.sample_surface_even(mesh_saddle, count=100000)
num_unif_samples = unif_samples.shape[0]
# print(unif_samples.shape)
```

1.3 Display 100k uniformly sampled points

[37]:
```
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(unif_samples)
draw_geometries([pcd])
plt.title("Saddle's 100K Uniformly Sampled Surface", fontweight ="bold")
```

[37]:
```
Text(0.5, 1.0, "Saddle's 100K Uniformly Sampled Surface")
```



1.4 4.2 Iterative farthest point sampling method (IFPS)

[38]:
```
start_time = time.time()

num_ifps_samples = 4000

# initialize an array for storing indices of remaining points
rem_pts_idx = np.arange(num_unif_samples)    # [0,1,2,...,99999]

# initialize an array for storing sampled/selected points indices
sampled_pts_idx = np.zeros(num_ifps_samples, dtype='int')  # (4000,)

# initialize distances to infinity
dists = np.ones_like(rem_pts_idx) * float('inf')  # (100000,)

# select a random point from unif_samples and add it to sampled points
selected_pt_idx = 0
# selected_pt_idx = np.random.randint(0, num_unif_samples)
sampled_pts_idx[0] = rem_pts_idx[selected_pt_idx]

# delete the selected point index from remaining points indices list
rem_pts_idx = np.delete(rem_pts_idx, selected_pt_idx)  # (99999,)

# iteratively select points from unif_samples and add it to sampled_pts
```

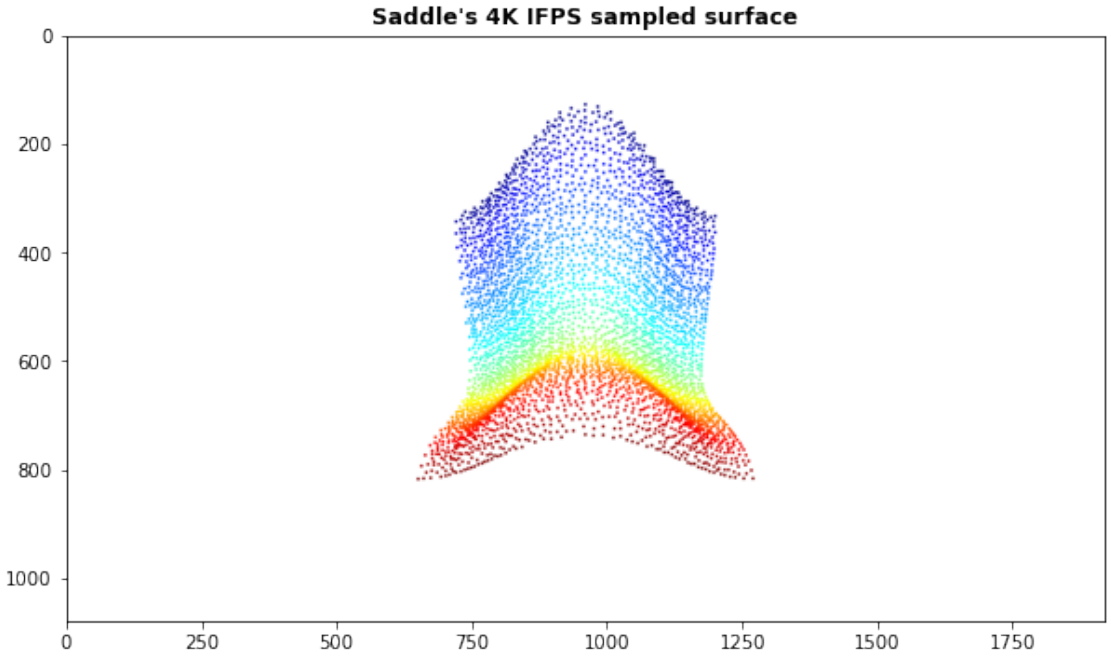
```
for i in tqdm(range(1, num_ifps_samples)):
    last_added_pt_idx = sampled_pts_idx[i-1]

    vec_to_last_added_pt = unif_samples[rem_pts_idx] -␣
 ↪unif_samples[last_added_pt_idx]
    dist_to_last_added_pt = np.linalg.norm(vec_to_last_added_pt, axis=1)
    dists[rem_pts_idx] = np.minimum(dist_to_last_added_pt, dists[rem_pts_idx])

    selected_pt_idx = np.argmax(dists[rem_pts_idx])
    sampled_pts_idx[i] = rem_pts_idx[selected_pt_idx]

    rem_pts_idx = np.delete(rem_pts_idx, selected_pt_idx)

ifps_samples = unif_samples[sampled_pts_idx]
print("IFPS sampled points shape:", ifps_samples.shape)

print("Time elapsed:", round(time.time() - start_time, 3), 's')
```

```
  0%|          | 0/3999 [00:00<?, ?it/s]
IFPS sampled points shape: (4000, 3)
Time elapsed: 33.632 s
```

1.5 Display the 4k point cloud sampled using IFPS method

[39]:
```
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(ifps_samples)
draw_geometries([pcd])
plt.title("Saddle's 4K IFPS sampled surface", fontweight ="bold")
```

[39]:
```
Text(0.5, 1.0, "Saddle's 4K IFPS sampled surface")
```



1.6 4.3 Estimation of normal vector at 4k sampled points using PCA

[42]:
```
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(ifps_samples)
geometry_lst = [pcd]

start_time = time.time()

num_NN_pts = 50
dists = np.zeros(num_unif_samples, dtype='float')
normal_vecs = np.zeros_like(ifps_samples)

# iterate through 4K sampled points and estimate normal at each point
for i in tqdm(range(num_ifps_samples)):
    pt_idx = sampled_pts_idx[i]
    sampled_pt = unif_samples[pt_idx]

    dists = np.linalg.norm(unif_samples - sampled_pt, axis=1)  # (100000,)
    idx_sorted_using_dists = np.argsort(dists)  # (100000,)
    assert(idx_sorted_using_dists[0] == pt_idx)

    NN_indices = idx_sorted_using_dists[1:num_NN_pts+1]  # (50,)

    # compute normal vector for each point using PCA
```

```
    x_mean = np.mean(unif_samples[NN_indices], axis=0)    # (3,)
    xi_minus_mean = (unif_samples[NN_indices] - x_mean)    # 50x3
    xi_cov_mat = np.matmul(xi_minus_mean.T, xi_minus_mean) / (num_NN_pts-1)   #␣
↪3x3

    eigenvals, eigenvecs = np.linalg.eig(xi_cov_mat)    # (3,), (3,3)
    normal_vec = eigenvecs[:, np.argmin(eigenvals)]
    normal_vec = normal_vec / np.linalg.norm(normal_vec)

    # orient normals so that they roughly point in the Y-direction
    if normal_vec[1] < 0:
        normal_vec *= -1
    normal_vecs[i] = normal_vec

    # drawing normals at every 50 iterations
    if (i % 25 == 0):
        arrow = create_arrow_from_vector(origin=sampled_pt, vector=normal_vec)
        geometry_lst.append(arrow)

print("Normal Vecs shape:", normal_vecs.shape)
print("Time elapsed:", round(time.time() - start_time, 3), 's')
```

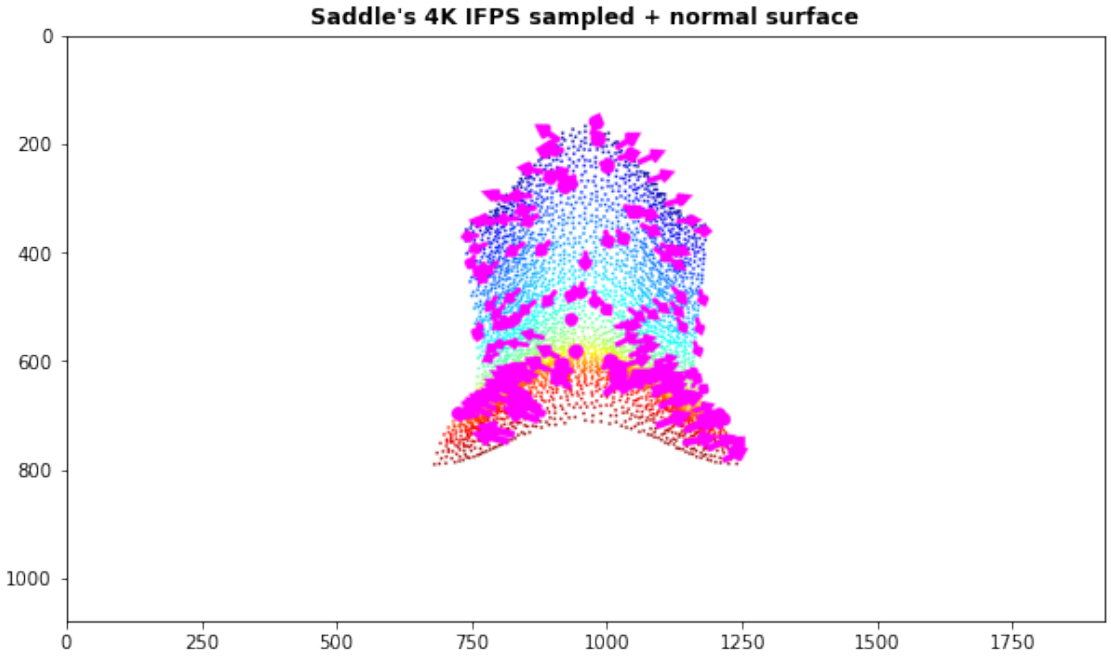
```
  0%|          | 0/4000 [00:00<?, ?it/s]
```

```
Normal Vecs shape: (4000, 3)
Time elapsed: 64.918 s
```

1.7 Plotted normal vectors for every 25 points for display

[43]:
```
draw_geometries(geometry_lst)
# open3d.visualization.draw_geometries(geometry_lst)
plt.title("Saddle's 4K IFPS sampled + normal surface", fontweight ="bold")
```

[43]:
```
Text(0.5, 1.0, "Saddle's 4K IFPS sampled + normal surface")
```



[58]:
```
# pcd = open3d.geometry.PointCloud()
# pcd.points = open3d.utility.Vector3dVector(ifps_samples)
# downpcd = open3d.geometry.voxel_down_sample(pcd, voxel_size=0.05)
# open3d.geometry.estimate_normals(downpcd, search_param=open3d.geometry.
 ↪KDTreeSearchParamHybrid(radius=0.1,max_nn=30))

# open3d.visualization.draw_geometries([downpcd])
```

1.8 4.4 Principal curvature estimation for point cloud (using modified Rusinkiewicz's method)

1.8.1 Algorithm

- Iteratively go over each point cloud in the 4k points.
- To apply Rusinkiewitz's method, we need faces using which we can define edges, face normals and vertex normals.
- For each point p, we sample nearby neighbouring points in the point cloud within an L2-ball of radius ϵ.
- For these sampled points and point p, we compute vector joining p to all the sampled points
- Next, for each vector v, we compute the dot products with other vectors in order to find two vectors in the sampled points which are most aligned with the vector v.
- Now, using vector v and two aligned vectors, form two triangles or two faces and store their vertex indices
- Keep repeating above steps for all the 4k sampled points to get mesh vertices and mesh faces

- Apply the standard Rubinkievitz's method on this mesh and compute the shape operator and hence the principal curvature for each faces
- For each point in the point cloud, compute its principal curvature by taking weighted mean of the principal curvatures of all the faces containing this point

[48]:
```
def compute_face_normals(mesh_vertices, mesh_faces):
    num_faces = mesh_faces.shape[0]
    num_vertices = mesh_vertices.shape[0]

    face_normals = []

    for itr in tqdm(range(num_faces)):
        vertices_idx = mesh_faces[itr]   # (3,)
        vertices = mesh_vertices[vertices_idx]   # 3x3

        # compute edge vectors for each face
        edges = np.zeros((3,3))
        edges[0] = vertices[2] - vertices[1]
        edges[1] = vertices[0] - vertices[2]
        edges[2] = vertices[1] - vertices[0]

        # compute face orthonormal basis vectors using Gram-Schmidt␣
↪orthogonalization
        Dfp = np.zeros((3,2))
        Dfp[:,0] = edges[0]
#         assert(np.dot(edges[0], edges[0]) - np.linalg.norm(edges[0])**2 <=␣
↪1e-10)
        proj_component = (np.dot(edges[1], edges[0]) / np.dot(edges[0],␣
↪edges[0])) * edges[0]
        Dfp[:,1] = edges[1] - proj_component
        Dfp = Dfp / np.linalg.norm(Dfp, axis=0)
#         assert(np.dot(Dfp[:,0], Dfp[:,1]) <= 1e-10)

        # compute face normal
        face_normal = np.cross(Dfp[:,0], Dfp[:,1])
        face_normals.append(face_normal)

    face_normals = np.array(face_normals)
    return face_normals
```

[53]:
```
def compute_shape_operator(mesh_vertices, mesh_faces):
    num_faces = mesh_faces.shape[0]
    num_vertices = mesh_vertices.shape[0]

    start_time = time.time()

    # compute face normals
```

```
    face_normals = compute_face_normals(mesh_vertices, mesh_faces)
#     face_normals = mesh.face_normals

    # compute vertex normals
    vertex_normals = trimesh.geometry.mean_vertex_normals(num_vertices,␣
 ↪mesh_faces, face_normals)
#     pcd.estimate_normals(open3d.geometry.KDTreeSearchParamKNN(knn=50))

    principal_curv = np.zeros((num_faces, 2))

    for itr in tqdm(range(num_faces)):
        vertices_idx = mesh_faces[itr]   # (3,)
        vertices = mesh_vertices[vertices_idx]   # 3x3

        # compute edge vectors for each face
        edges = np.zeros((3,3))
        edges[0] = vertices[2] - vertices[1]
        edges[1] = vertices[0] - vertices[2]
        edges[2] = vertices[1] - vertices[0]

        # compute face orthonormal basis vectors using Gram-Schmidt
        Dfp = np.zeros((3,2))
        Dfp[:,0] = edges[0]
        proj_component = (np.dot(edges[1], edges[0]) / np.dot(edges[0],␣
 ↪edges[0])) * edges[0]
        Dfp[:,1] = edges[1] - proj_component
        Dfp = Dfp / np.linalg.norm(Dfp, axis=0)
        assert(np.dot(Dfp[:,0], Dfp[:,1]) <= 1e-10)

        lhs1 = np.matmul(Dfp.T, edges[0])
        rhs1 = np.matmul(Dfp.T, vertex_normals[vertices_idx[2]] -␣
 ↪vertex_normals[vertices_idx[1]])

        lhs2 = np.matmul(Dfp.T, edges[1])
        rhs2 = np.matmul(Dfp.T, vertex_normals[vertices_idx[0]] -␣
 ↪vertex_normals[vertices_idx[2]])

        lhs3 = np.matmul(Dfp.T, edges[2])
        rhs3 = np.matmul(Dfp.T, vertex_normals[vertices_idx[1]] -␣
 ↪vertex_normals[vertices_idx[0]])

        A = np.array([lhs1, lhs2, lhs3]).T
        B = np.array([rhs1, rhs2, rhs3]).T

        S = np.matmul(np.matmul(B, A.T), np.linalg.inv(np.matmul(A, A.T)))

        principal_curv[itr], _ = np.linalg.eig(S)
```

```
    print("Time taken:", round(time.time() - start_time, 3), 's')

    return principal_curv
```

[51]:
```
eps_radius = 0.1

mesh_faces = []
mesh_vertices = []

for itr in tqdm(range(num_ifps_samples)):
    sampled_pt = ifps_samples[itr]    # (3,)
    mesh_vertices.append(sampled_pt)

    dists = np.linalg.norm(ifps_samples - sampled_pt, axis=1)   # (4000,)
    idx_sorted_dists = np.argsort(dists)   # (4000,)
#     plt.hist(dists, bins='auto')

    nn_pts_idx = np.where(np.logical_and(dists > 0, dists < eps_radius))[0]  #␣
 ↪(n,)
    nn_pts = ifps_samples[nn_pts_idx]   # (n, 3)
    num_nn_pts = nn_pts.shape[0]        # =n

    triangle_counter = np.zeros(num_nn_pts)

    # compute the vectors joining central point to the neigh points
    center_to_nn_vec = nn_pts - sampled_pt   # (n,3)
    center_to_nn_vec_len = np.linalg.norm(center_to_nn_vec, axis=1,␣
 ↪keepdims=True)  # (n,1)
    center_to_nn_vec = center_to_nn_vec / center_to_nn_vec_len   # (n,3)

    for j in range(num_nn_pts):
        # compute dot product
        dot_prod = np.dot(center_to_nn_vec, center_to_nn_vec[j])   # (n,)
        dot_prod_sort_idx = np.argsort(dot_prod)[::-1]   # decreasing order  ␣
 ↪(n,)

#         tri_cand_pts = nn_pts[dot_prod_sort_idx[1:3]]   # (2,3)
        mesh_faces.append([itr, nn_pts_idx[j],␣
 ↪nn_pts_idx[dot_prod_sort_idx[1]]])
        mesh_faces.append([itr, nn_pts_idx[j],␣
 ↪nn_pts_idx[dot_prod_sort_idx[2]]])
```

```
  0%|          | 0/4000 [00:00<?, ?it/s]
```

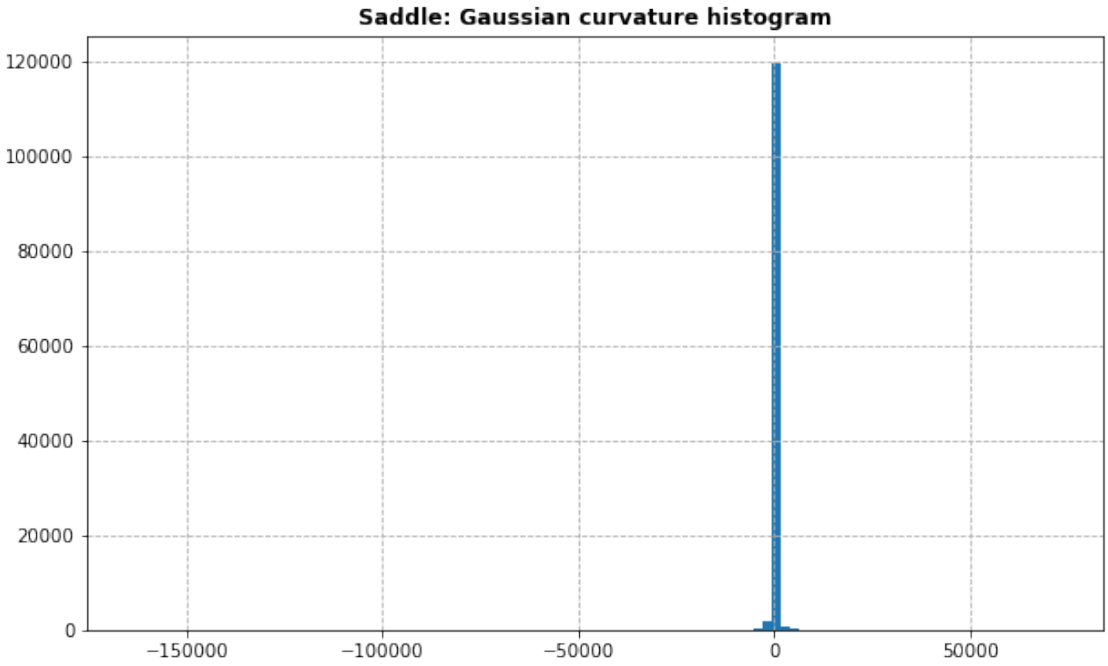
[54]:
```
mesh_vertices = np.array(mesh_vertices)
mesh_faces = np.array(mesh_faces)
```

```
print(mesh_vertices.shape, mesh_faces.shape)

principal_curv_saddle = compute_shape_operator(mesh_vertices, mesh_faces)
```

```
(4000, 3) (122732, 3)
  0%|          | 0/122732 [00:00<?, ?it/s]
  0%|          | 0/122732 [00:00<?, ?it/s]
<ipython-input-53-8b7665cfbc14>:49: ComplexWarning: Casting complex values to
real discards the imaginary part
  principal_curv[itr], _ = np.linalg.eig(S)
Time taken: 35.767 s
```

[57]:
```
print(principal_curv_saddle.shape)
print(principal_curv_saddle)

gauss_curv_saddle = np.multiply(principal_curv_saddle[:,0],␣
 ↪principal_curv_saddle[:,1])

fig = plt.figure(figsize=(10,6))
# plt.subplot(1,2,1)
plt.hist(gauss_curv_saddle, 100)
plt.title("Saddle: Gaussian curvature histogram", fontweight ="bold")
plt.grid(linestyle='--')
```

```
(122732, 2)
[[-8.91137050e-02 -2.42768841e+00]
 [ 8.11475587e-02 -1.48275232e+00]
 [-3.80652195e-01 -3.80652195e-01]
 …
 [-2.51951248e+00  1.52251743e+01]
 [-8.60733979e-01 -1.36736027e+02]
 [-4.07692794e+00 -4.07692794e+00]]
```



2 Part V: Course Feedback

2.0.1 1. How many hours did you spend on this homework?

Answer: Roughly between 30-40 hrs

2.0.2 2. How many hours did you spend on the course each week?

Answer: Initial weeks - approx. 10-15 hrs/week, Last 2 weeks - approx. 30-40 hrs/week

2.0.3 3. Do you have any course related feedback?

Answer: - The assignment was a bit time consuming compared to other course's assignments

2.1 References

- https://minibatchai.com/sampling/2021/08/07/FPS.html
- https://trimsh.org/trimesh.geometry.html
-

2.2 Acknowledgements

I had discussion about some of the problems while doing this assignment with my following classmates: - Chinmay Talegoankar - Sambaran Ghosal - Srinidhi