# HW1_sol

February 5, 2021

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

## 1 1 Rotation

1.1 Your answer here

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

$$|\frac{p+q}{2}| = \frac{\sqrt{3}}{2}$$

$$r = \frac{\sqrt{6}}{3} + \frac{\sqrt{6}}{6}i + \frac{\sqrt{6}}{6}j$$

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

Axis:

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

Angle:

$$70.5°$$

1.2 your answer here

p:

$$\begin{bmatrix} \frac{\pi}{2} & 0 & 0 \end{bmatrix}$$

q:

$$\begin{bmatrix} 0 & \frac{\pi}{2} & 0 \end{bmatrix}$$

1.3.a

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

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

$$\exp([\omega_p]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\exp([\omega_q]) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

1.3.b your answer here

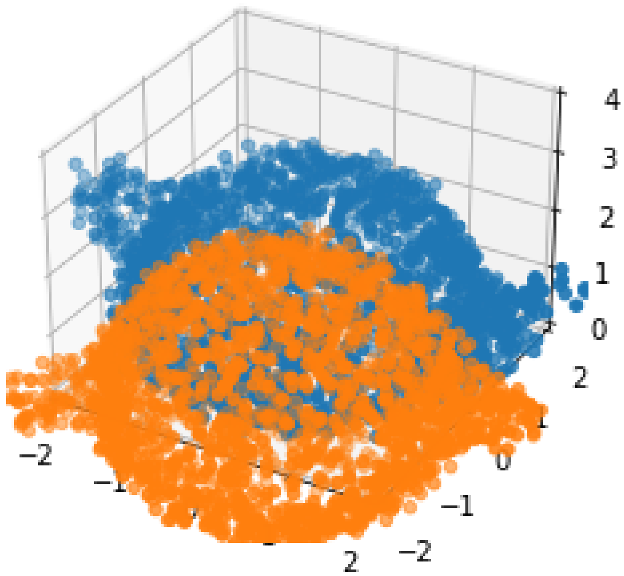
$$\exp([\omega_p] + [\omega_q]) = \begin{bmatrix} 1/2 + \cos(\pi/\sqrt{2})/2 & 1/2 - \cos(\pi/\sqrt{2})/2 & \sin(\pi/\sqrt{2})/\sqrt{2} \\ 1/2 - \cos(\pi/\sqrt{2})/2 & 1/2 + \cos(\pi/\sqrt{2})/2 & -(\sin(\pi/\sqrt{2})/\sqrt{2}) \\ -(\sin(\pi/\sqrt{2})/\sqrt{2}) & \sin(\pi/\sqrt{2})/\sqrt{(2)} & \cos(\pi/\sqrt{2}) \end{bmatrix}$$

$$\exp([\omega_p])\exp([\omega_q]) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

1.3.c your code here

[2]:
```
# Note Matplotlib is only suitable for simple 3D visualization.
# For later problems, you should not use Matplotlib to do the plotting
from mpl_toolkits.mplot3d import Axes3D
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()
    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('HW1_P1.npz')
X = npz['X']
Y = npz['Y']
compare_points(X, Y)  # noisy teapotsand
```




[4]:
```
np.savez('HW1_P1.npz', X=Y, Y=X)
```

[5]:
```
# copy-paste your hw0 solve module here
def hw0_solve(A, b, eps):
    x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)

    # case 1
    if x @ x < eps:
        return x

    # case 2
    d, U = np.linalg.eigh(A.T@A)  # SVD of A may be faster
    k = U.T@(A.T@b)

    def func(lam):
        return ((k / (d + 2 * lam))**2).sum() - eps

    # find a valid pair func(a) > 0, func(b) < 0
    lo = 0
    hi = 1
    while func(hi) > 0:
        lo, hi = hi, hi * 2
    # bisect
    thres = 1e-12
    while True:
        mi = (lo+hi) / 2
```

```
        v = func(mi)
        if abs(v) < thres:
            break
        if v > 0:
            lo = mi
        else:
            hi = mi
    lam = mi
    x = U@(np.diag(1/(d + 2 * lam))@(U.T@(A.T@b)))
    return x
```

[9]:

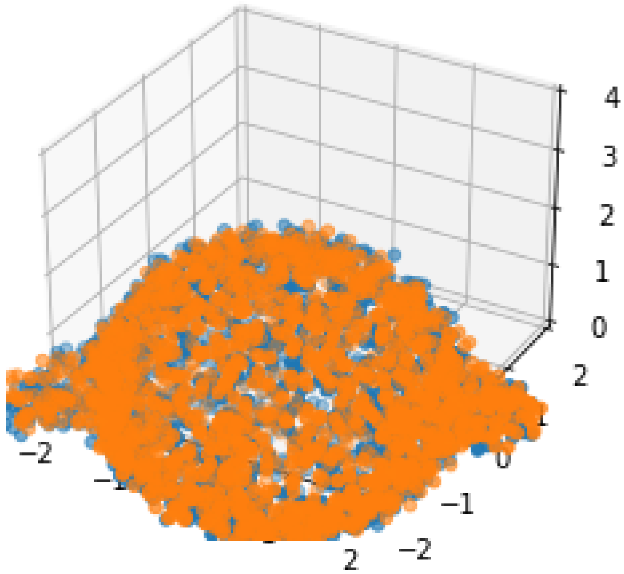
```
R1 = np.eye(3)
# solve this problem here, and store your final results in R1
for __ in range(100):
    A = R1[:, [2,0,1], None] * X[[1,2,0]][None] - R1[:, [1,2,0], None] *␣
 ↪X[[2,0,1]][None]
    A = A.transpose(2, 0, 1)
    A = A.reshape((-1, A.shape[-1]))
    b = Y - R1@X
    b = b.T.reshape(-1)
    x = hw0_solve(A, b, 0.1)

    theta = np.linalg.norm(x)
    d = x / theta
    a1,a2,a3 = d
    K = np.array([[0, -a3, a2], [a3, 0, -a1], [-a2, a1, 0]])
    exp = np.eye(3) + np.sin(theta) * K + (1-np.cos(theta)) * (K@K)
    R1 = R1 @ exp
```

[10]:

```
# Testing code, you should see the blue and orange points roughly overlap
compare_points(R1@X, Y)
R1.T@R1
```

[10]:

```
array([[ 1.00000000e+00, -9.81665362e-17,  2.23582699e-16],
       [-9.81665362e-17,  1.00000000e+00, -2.02566755e-17],
       [ 2.23582699e-16, -2.02566755e-17,  1.00000000e+00]])
```




1.4.a your solution here

p'=-p:

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

q'=-q:

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

The Exponential coordinates for $p, -p$ represent the same rotation. The same holds for $q, -q$.

For any quaternion $r$, $(r, -r)$ represents the same rotation.

Let $r = (w, x, y, z) = \cos\frac{\theta}{2} + \mathbf{u}\sin\frac{\theta}{2}$,

$$-r = -\cos\frac{\theta}{2} - \mathbf{u}\sin\frac{\theta}{2} = \cos(\pi - \frac{\theta}{2}) + (-\mathbf{u})\sin(\pi - \frac{\theta}{2}) = \cos(\frac{2\pi - \theta}{2}) + (-\mathbf{u})\sin(\frac{2\pi - \theta}{2})$$

It represents rotating around $-\mathbf{u}$ for angle $2\pi - \theta$, which is equivalent to rotating around $\mathbf{u}$ for angle $\theta$.

1.5.b your solution here

No, since $r$ and $-r$ represents the same rotation (thus the distance should be the smallest) but the L2 distance is the largest.

## 2 Geometry

2(Warm up)

Equation for ellipsoid:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

Plug in $f(\theta, \phi)$ we find this equation is satisfied.

2.1 your solution here

Just make sure **p** is a point, **v** is a vector, $\gamma$ is a straight line passing through **p** with direction **v**, $f \circ \gamma$ is on the ellipsoid.

2.2 your solution here

[11]:
```
a, b, c = 1, 1, 0.5
```

[12]:
```
# 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
import open3d
import numpy as np
import matplotlib.pyplot as plt

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)
```

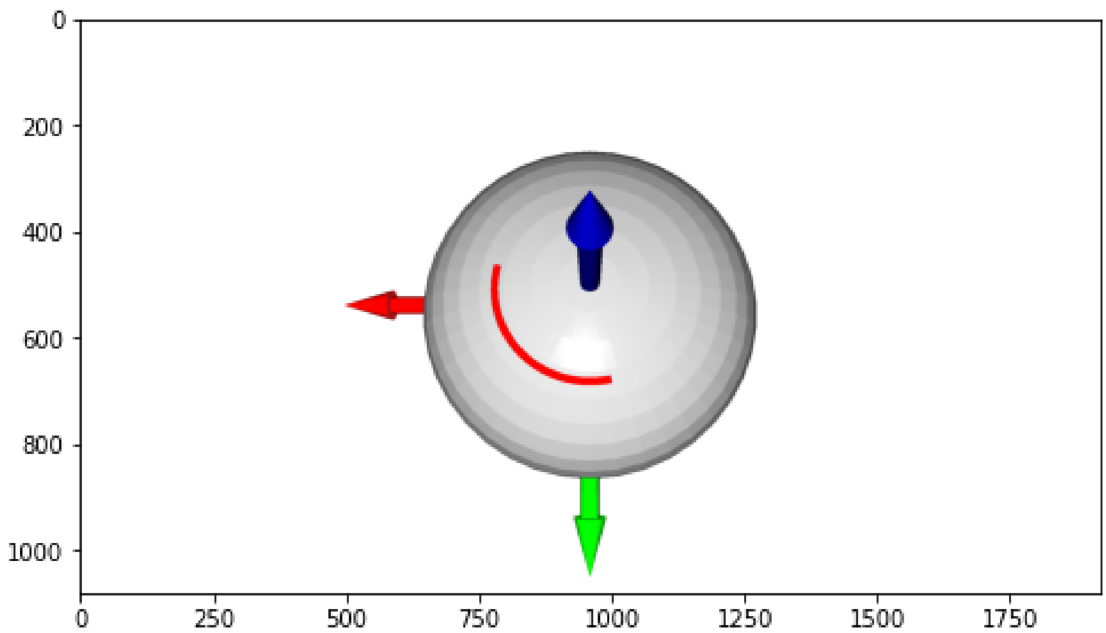
```python
    arrow = open3d.geometry.TriangleMesh.create_arrow(0.05, 0.1, 0.25, 0.2)
    arrow.paint_uniform_color([1,0,1])
    T = np.eye(4)
    T[:3, 3] = np.array(origin)
    T[:3,:3] = axangle2mat(np.cross(z, v), angle)
    arrow.transform(T)
    return arrow

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
```

[13]:
```python
def f(uv):
    if uv.shape == (2,):
        return np.stack([a * np.cos(uv[..., 0]) * np.sin(uv[..., 1]),
                        b * np.sin(uv[..., 0]) * np.sin(uv[..., 1]),
                        c * np.cos(uv[..., 1])])
    return np.stack([a * np.cos(uv[..., 0]) * np.sin(uv[..., 1]),
                    b * np.sin(uv[..., 0]) * np.sin(uv[..., 1]),
                    c * np.cos(uv[..., 1])], 1)
```

[14]:
```python
p = np.array([np.pi/4,np.pi/6])
v = np.array([1,0])
```

```
[15]: t = np.linspace(-1, 1, 1000)
curve = p + t[:, None] * v
x, y, z = f(curve).T
points = np.stack([x,y,z],1)
lines = [[i,i+1] for i in range(len(points)-1)]
```

```
[16]: ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
curve = create_lines(points)
draw_geometries([ellipsoid, cf] + curve)
```




2.3.a your computation here

$$(f \circ \gamma)'(t) = f'(\gamma(t))\gamma'(t) = \begin{bmatrix} -\sin u \sin v & \cos u \cos v \\ \cos u \sin v & \sin u \cos v \\ 0 & -\frac{1}{2}\sin v \end{bmatrix} \mathbf{v}$$

$$Df_{\mathbf{p}} = \begin{bmatrix} -\sin u \sin v & \cos u \cos v \\ \cos u \sin v & \sin u \cos v \\ 0 & -\frac{1}{2}\sin v \end{bmatrix}$$

```
[17]: def Df(uv):
    assert uv.shape == (2,)
    u, v = uv
    Df = np.stack([[-a * np.sin(u) * np.sin(v), a * np.cos(u) * np.cos(v)],
```

```
        [b * np.cos(u) * np.sin(v), b * np.sin(u) * np.cos(v)],
        [0, -c * np.sin(v)]])
    return Df

Df(p) @ v
```

[17]: array([-0.35355339,  0.35355339,  0.        ])

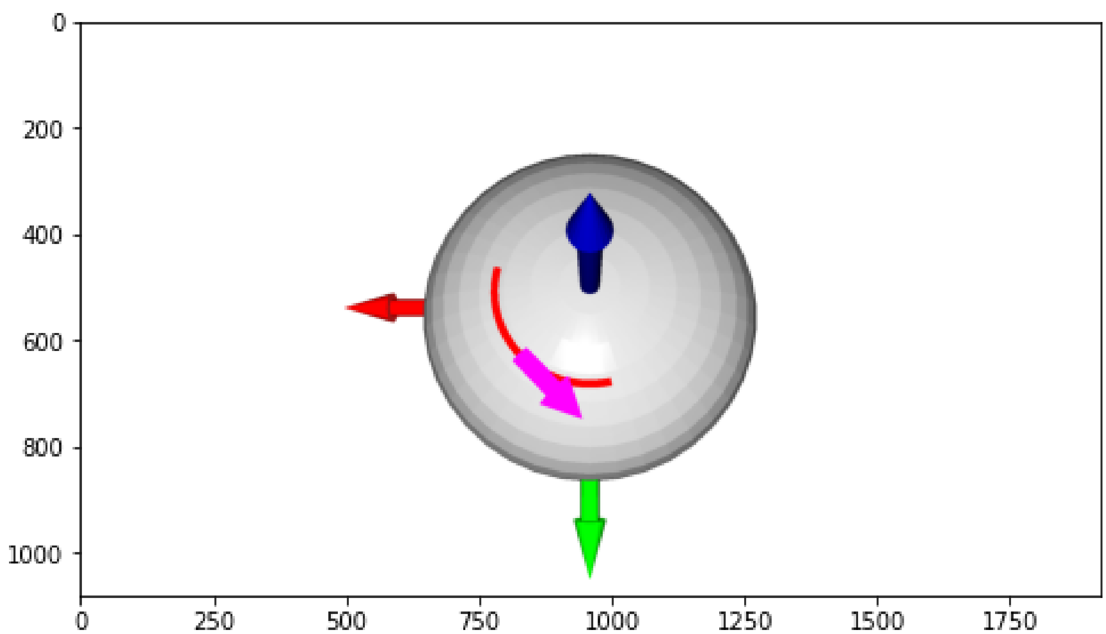
2.3.b

$Df_{\mathbf{p}}$ maps the tangent plane in 2D to the tangent plane in 3D

2.3.c your plot here

[18]:
```
from transforms3d.axangles import axangle2mat
```

[19]:
```
arrow = create_arrow_from_vector(f(p), Df(p) @ v)

ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
curve = create_lines(points)
draw_geometries([ellipsoid, cf, arrow] + curve)
```




2.3.b

$$\gamma(t) = \mathbf{p} + \mathbf{v}t = [2, 1]^T + [1, 0]^T t$$

$$Df_{\mathbf{p}}(\mathbf{v}) = f'(\gamma(t))\gamma'(t)|_{t=0} = f'(\mathbf{p})\mathbf{v}$$

$$Df_{\mathbf{p}} = \begin{bmatrix} -\sin u \sin v & \cos u \cos v \\ \cos u \sin v & \sin u \cos v \\ 0 & -\frac{1}{2}\sin v \end{bmatrix}$$

[20]:
```
normal = np.cross(Df(p)[:,0], Df(p)[:,1])
normal /= np.linalg.norm(normal)
normal
```

[20]:
```
array([-0.19611614, -0.19611614, -0.96076892])
```

2.3.e

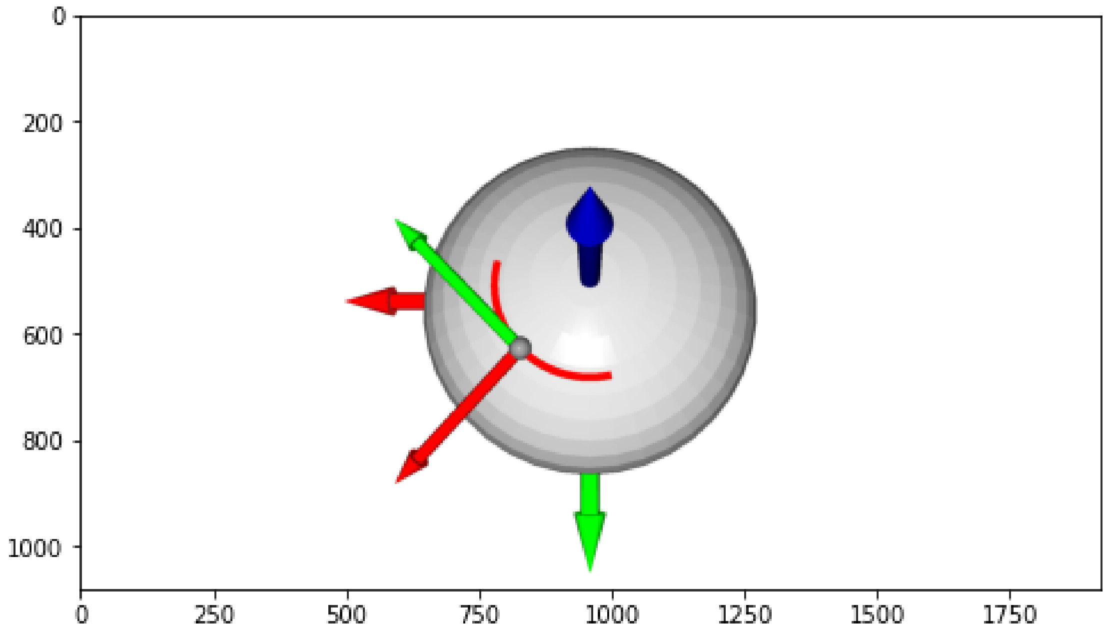
[21]:
```
X = Df(p)[:,1]
X /= np.linalg.norm(X)
Y = np.cross(normal, X)
onb = np.array([X, Y, normal]).T
```

2.3.f

[22]:
```
onb
```

[22]:
```
array([[ 0.67936622,  0.70710678, -0.19611614],
       [ 0.67936622, -0.70710678, -0.19611614],
       [-0.2773501 ,  0.        , -0.96076892]])
```

[23]:
```
frame = open3d.geometry.TriangleMesh.create_coordinate_frame()
T = np.eye(4)
T[:3,:3] = onb
T[:3,3] = f(p)
frame.transform(T)

ellipsoid = create_ellipsoid(a, b, c)
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
curve = create_lines(points)
draw_geometries([ellipsoid, cf, frame] + curve)
```




2.4.a

$$g'_{\mathbf{v}}(t) = f'(\gamma(t))\mathbf{v} = [-\sin u(t)\sin v(t), \cos u(t)\sin v(t), 0]^T$$
$$||g'_{\mathbf{v}}(t)|| = \sin(v(t)) = \sin(\pi/6) = 1/2$$
$$s(t) = \int_0^t ||g'_{\mathbf{v}}(t)||dt = \frac{1}{2}t$$

2.4.b

$$h_{\mathbf{v}}(s) = g_{\mathbf{v}}(t(s)) = g_{\mathbf{v}}(2s) = f(\gamma(2s)) = [\frac{1}{2}\cos(\frac{\pi}{4}+2s), \frac{1}{2}\sin(\frac{\pi}{4}+2s), \frac{\sqrt{3}}{4}]^T$$

2.4.c

$$T(s) = h'_{\mathbf{v}}(s) = [-\sin(\frac{\pi}{4}+2s), \cos(\frac{\pi}{4}+2s), 0]^T$$
$$T'(s) = [-2\cos(\frac{\pi}{4}+2s), -2\sin(\frac{\pi}{4}+2s), 0]^T$$
$$N(s) = \frac{T'(s)}{||T'(s)||} = [-\cos(\frac{\pi}{4}+2s), -\sin(\frac{\pi}{4}+2s), 0]^T$$

2.4.d

The curve normal is different from the surface normal

2.5.a

We find the normal by

$$N_0 = \begin{bmatrix} -\sin u \sin v \\ \cos u \sin v \\ 0 \end{bmatrix} \times \begin{bmatrix} \cos u \cos v \\ \sin u \cos v \\ -\frac{1}{2}\sin v \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}\cos u \sin^2 v \\ -\frac{1}{2}\sin u \sin^2 v \\ -\sin v \cos v \end{bmatrix}$$

$$N = \frac{N_0}{||N_0||} = \frac{-1}{\sqrt{\sin^2 v + 4\cos^2 v}} \begin{bmatrix} \cos u \sin v \\ \sin u \sin v \\ 2\cos v \end{bmatrix}$$

2.5.b

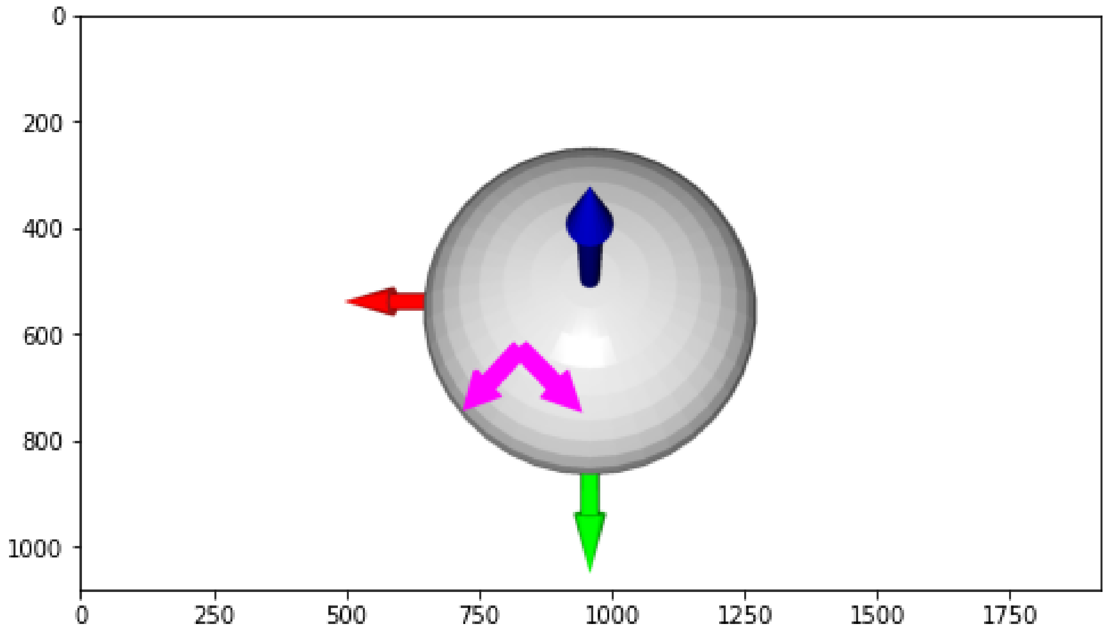
$$DN_p = \begin{bmatrix} \frac{\sin u \sin v}{\sqrt{\sin^2 v + 4\cos^2 v}} & -\frac{4\cos(u)\cos(v)}{(4\cos^2(v)+\sin^2(v))^{3/2}} \\ -\frac{\cos u \sin v}{\sqrt{\sin^2 v + 4\cos^2 v}} & -\frac{4\cos(v)\sin(u)}{(4\cos^2(v)+\sin^2(v))^{3/2}} \\ 0 & \frac{2\sin(v)}{(4\cos^2(v)+\sin^2(v))^{3/2}} \end{bmatrix}$$

$$Df_{\mathbf{p}} = \begin{bmatrix} -\sin u \sin v & \cos u \cos v \\ \cos u \sin v & \sin u \cos v \\ 0 & -\frac{1}{2}\sin v \end{bmatrix}$$

Since $DN_{\mathbf{p}} = Df_{\mathbf{p}}S$, comparing the columns of $DN_{\mathbf{p}}$ and $Df_{\mathbf{p}}$, we can see the shape operator is diagonal. Therefore its eigenvectors are $[0, 1], [1, 0]$.

2.5.c

[24]:
```
arrow1 = create_arrow_from_vector(f(p), Df(p)[:,0])
arrow2 = create_arrow_from_vector(f(p), Df(p)[:,1])
cf = open3d.geometry.TriangleMesh.create_coordinate_frame()
cf.scale(1.5, (0,0,0))
ellipsoid = create_ellipsoid(a, b, c)
draw_geometries([ellipsoid, arrow1, arrow2, cf])
```




2.5.d Longitude, Latitude

3.1 your proof here

Let the surface normal at $\mathbf{p}$ be $n_{\mathbf{p}}$,

$$M_{\mathbf{p}}n_{\mathbf{p}} = \frac{1}{2\pi}\int_{-\pi}^{\pi}\kappa_\theta t_\theta t_\theta^T n_{\mathbf{p}} d\theta = \frac{1}{2\pi}\int_{-\pi}^{\pi}\kappa_\theta t_\theta \cdot 0 d\theta = \mathbf{0} = 0n_{\mathbf{p}}$$

The normal is an eigenvector corresponding to eigenvalue 0.

3.2 your proof here

Let $T_1, T_2$ be the principal curvature directions.

$$\begin{aligned}
M_{\mathbf{p}}T_1 &= \frac{1}{2\pi}\int_{-\pi}^{\pi}\kappa_\theta t_\theta t_\theta^T T_1 d\theta && (1)\\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi}\kappa_\theta t_\theta (cos\theta T_1 + \sin\theta T_2)^T T_1 d\theta && (2)\\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi}\kappa_\theta t_\theta cos\theta d\theta && (3)\\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi}(\kappa_1\cos^2\theta + \kappa_2\sin^2\theta)(\cos\theta T_1 + \sin\theta T_2)cos\theta d\theta && (4)\\
&= \frac{1}{2\pi}(\frac{3\pi}{4}\kappa_1 T_1 + \frac{\pi}{4}\kappa_2 T_1) && (5)\\
&= (\frac{3}{8}\kappa_1 + \frac{1}{8}\kappa_2)T_1 && (6)
\end{aligned}$$

Similar derivation holds for $T_2$.

3.3 your solution here

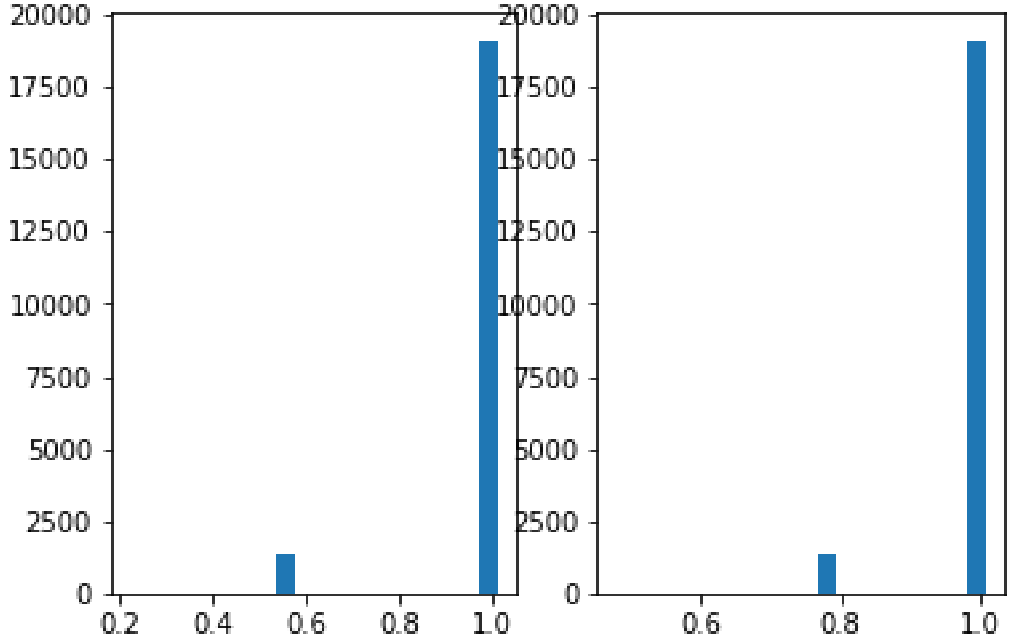
[25]:
```
from tqdm import tqdm
import trimesh
import numpy as np
```

[26]:
```
mesh = trimesh.load('icosphere.obj')
c1 = []
c2 = []
fv = []
for f_id, (v0,v1,v2) in enumerate(tqdm(mesh.faces)):
    p0 = mesh.vertices[v0]
    p1 = mesh.vertices[v1]
    p2 = mesh.vertices[v2]
    n0 = mesh.vertex_normals[v0]
    n1 = mesh.vertex_normals[v1]
    n2 = mesh.vertex_normals[v2]
    n = mesh.face_normals[f_id]

    fv.append((p0 + p1 + p2) / 3)

    X = np.array([1,0,0])
    if abs(n@X) > 0.95:
        xu = np.cross(n, X)
    else:
        xu = np.cross(n, np.array([0,1,0]))
    xu /= np.linalg.norm(xu)
    xv = np.cross(n, xu)

    e0 = p2-p1
    e1 = p0-p2
    e2 = p1-p0

    DfT = np.array([xu, xv])
    r0 = DfT @ (n2-n1)
    r1 = DfT @ (n0-n2)
    r2 = DfT @ (n1-n0)

    l0 = DfT @ e0
    l1 = DfT @ e1
    l2 = DfT @ e2

    A = np.zeros([6, 4])
    A[:3, :2] = np.array([l0, l1, l2])
    A[3:, 2:] = np.array([l0, l1, l2])
    b = np.array([r0, r1, r2]).T.reshape(-1)
    S = np.linalg.lstsq(A, b)[0].reshape((2,2))
```

```python
    eig, ev = np.linalg.eigh(S)
    c1.append(eig[0])
    c2.append(eig[1])
c1 = np.array(c1)
c2 = np.array(c2)
fv = np.array(fv)

gc = c1 * c2
mc = (c1+c2)/2
import matplotlib.pyplot as plt
plt.subplot(1, 2, 1)
plt.hist(gc, bins=20)
plt.subplot(1, 2, 2)
plt.hist(mc, bins=20)
```

```
WARNING - 2021-02-05 12:27:46,340 - obj - unable to load materials from:
icosphere.mtl
  0%|          | 0/20480 [00:00<?, ?it/s]<ipython-input-26-b26f3d708d81>:41:
FutureWarning: `rcond` parameter will change to the default of machine precision
times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  S = np.linalg.lstsq(A, b)[0].reshape((2,2))
100%|     | 20480/20480 [00:08<00:00, 2323.32it/s]
```

[26]:
```
(array([1.0000e+01, 0.0000e+00, 1.0000e+01, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 4.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 1.3230e+03, 1.0000e+00, 0.0000e+00, 4.0000e+00,
        0.0000e+00, 0.0000e+00, 1.6000e+01, 0.0000e+00, 1.9112e+04]),
 array([0.47771811, 0.50415684, 0.53059557, 0.55703431, 0.58347304,
        0.60991177, 0.6363505 , 0.66278923, 0.68922797, 0.7156667 ,
        0.74210543, 0.76854416, 0.7949829 , 0.82142163, 0.84786036,
        0.87429909, 0.90073783, 0.92717656, 0.95361529, 0.98005402,
        1.00649275]),
 <BarContainer object of 20 artists>)
```




[27]:

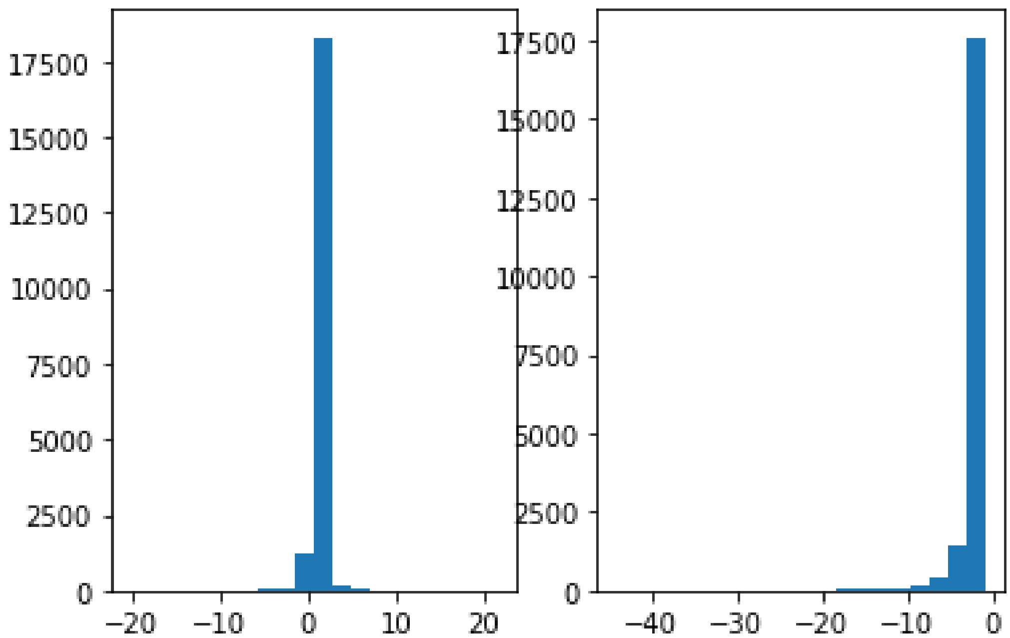
```
mesh = trimesh.load('sievert.obj')
c1 = []
c2 = []
fv = []
for f_id, (v0,v1,v2) in enumerate(tqdm(mesh.faces)):
    p0 = mesh.vertices[v0]
    p1 = mesh.vertices[v1]
    p2 = mesh.vertices[v2]
    n0 = mesh.vertex_normals[v0]
    n1 = mesh.vertex_normals[v1]
    n2 = mesh.vertex_normals[v2]
    n = mesh.face_normals[f_id]

    fv.append((p0 + p1 + p2) / 3)

    X = np.array([1,0,0])
    if abs(n@X) > 0.95:
        xu = np.cross(n, X)
    else:
        xu = np.cross(n, np.array([0,1,0]))
    xu /= np.linalg.norm(xu)
    xv = np.cross(n, xu)

    e0 = p2-p1
    e1 = p0-p2
```

```
    e2 = p1-p0

    DfT = np.array([xu, xv])
    r0 = DfT @ (n2-n1)
    r1 = DfT @ (n0-n2)
    r2 = DfT @ (n1-n0)

    l0 = DfT @ e0
    l1 = DfT @ e1
    l2 = DfT @ e2

    A = np.zeros([6, 4])
    A[:3, :2] = np.array([l0, l1, l2])
    A[3:, 2:] = np.array([l0, l1, l2])
    b = np.array([r0, r1, r2]).T.reshape(-1)
    S = np.linalg.lstsq(A, b)[0].reshape((2,2))
    eig, ev = np.linalg.eigh(S)
    c1.append(eig[0])
    c2.append(eig[1])
c1 = np.array(c1)
c2 = np.array(c2)
fv = np.array(fv)

gc = c1 * c2
mc = (c1+c2)/2
import matplotlib.pyplot as plt
plt.subplot(1, 2, 1)
plt.hist(gc, bins=20)
plt.subplot(1, 2, 2)
plt.hist(mc, bins=20)
```

```
  0%|          | 0/20000 [00:00<?, ?it/s]<ipython-input-27-a69ad0e45cc8>:41:
FutureWarning: `rcond` parameter will change to the default of machine precision
times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  S = np.linalg.lstsq(A, b)[0].reshape((2,2))
100%|     | 20000/20000 [00:08<00:00, 2342.55it/s]
```

[27]:
```
(array([2.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 4.0000e+00,
        0.0000e+00, 8.0000e+00, 2.0000e+00, 6.0000e+00, 1.2000e+01,
        1.4000e+01, 1.6000e+01, 2.6000e+01, 3.6000e+01, 6.4000e+01,
        1.1000e+02, 1.8600e+02, 4.4800e+02, 1.4480e+03, 1.7618e+04]),
 array([-44.1839162 , -42.02474829, -39.86558039, -37.70641248,
        -35.54724458, -33.38807667, -31.22890877, -29.06974087,
        -26.91057296, -24.75140506, -22.59223715, -20.43306925,
        -18.27390134, -16.11473344, -13.95556553, -11.79639763,
```

```
        -9.63722973,  -7.47806182,  -5.31889392,  -3.15972601,
        -1.00055811]),
 <BarContainer object of 20 artists>)
```




4.1 your solution here

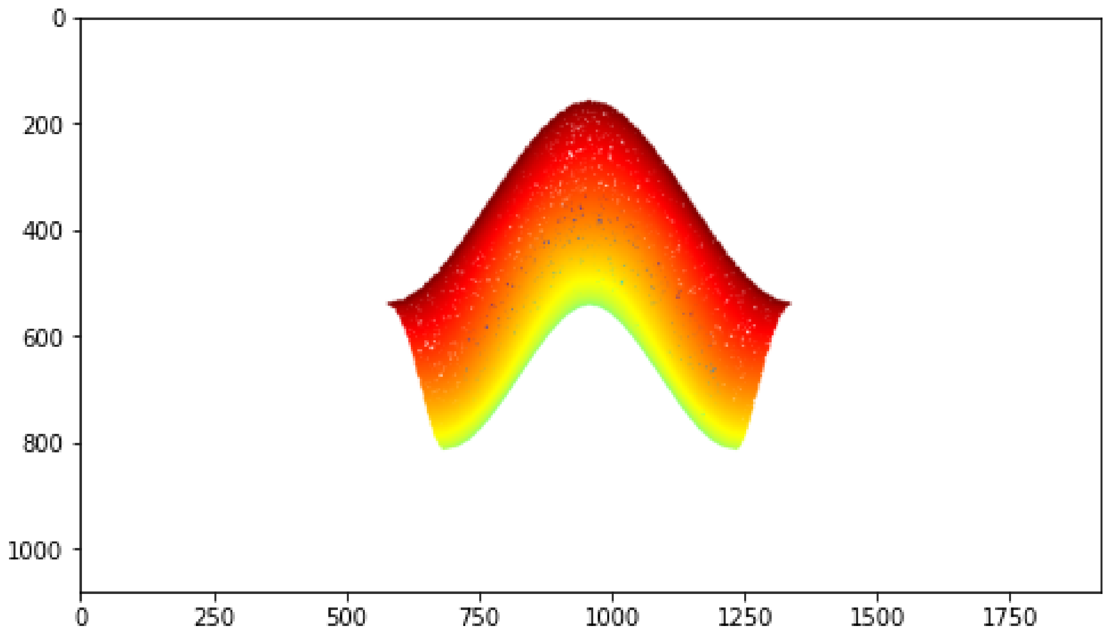
[28]:
```
import open3d
import numpy as np
import matplotlib.pyplot as plt

vis = open3d.visualization.Visualizer()
vis.create_window(visible = False)
opt = vis.get_render_option()
opt.point_show_normal = True

def draw_geometries(geoms):
    for g in geoms:
        vis.add_geometry(g)
    view_ctl = vis.get_view_control()
    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)
```

[29]:
```python
import trimesh
mesh = trimesh.load('saddle.obj')
points = trimesh.sample.sample_surface(mesh, 100000)[0]
```

```
INFO - 2021-02-05 12:28:04,211 - base - triangulating quad faces
```

[30]:
```python
import open3d
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(points)
draw_geometries([pcd])
```




4.2 your solution here

[31]:
```python
points = points.astype(np.float32)
new_points = np.zeros((4000, 3), dtype=np.float32)
```

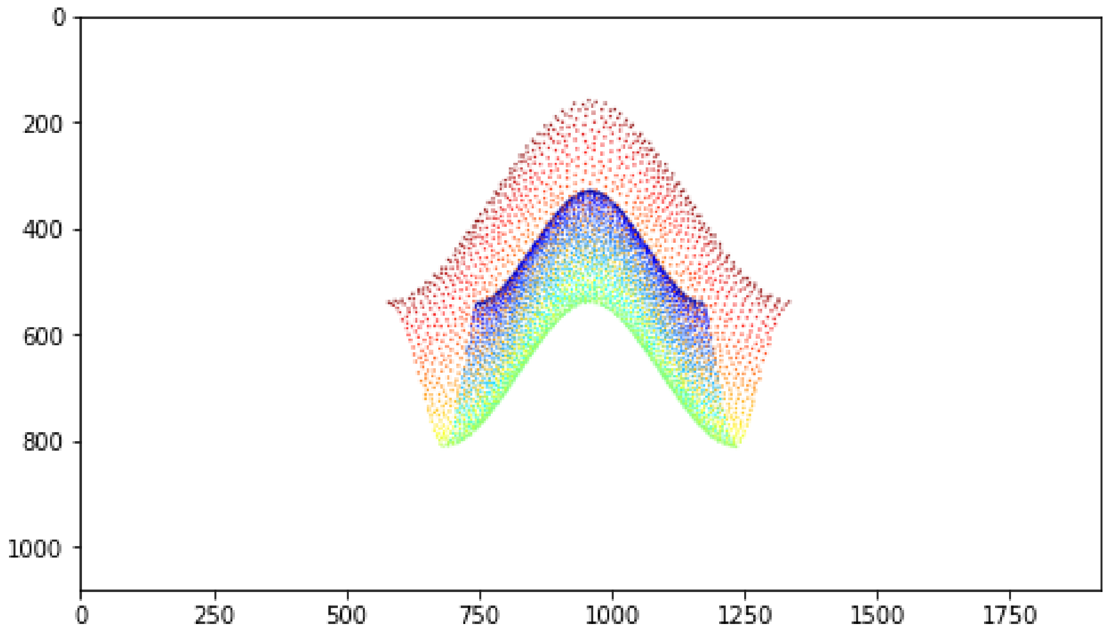
[32]:
```python
from tqdm import trange
```

[33]:
```python
new_points = np.zeros((4000, 3), dtype=np.float32)
dist = np.ones(points.shape[0], dtype=np.float32) * np.inf
for i in trange(4000):
    # pick point with max dist
    idx = np.argmax(dist)
    new_points[i] = points[idx]
```

```
    new_dist = ((points - points[idx]) ** 2).sum(-1)
    dist = np.minimum(dist, new_dist)
new_points = np.array(new_points)
```

100%|          | 4000/4000 [00:08<00:00, 464.48it/s]

[34]:
```
import open3d
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(new_points)
draw_geometries([pcd])
```




4.3 your solution here

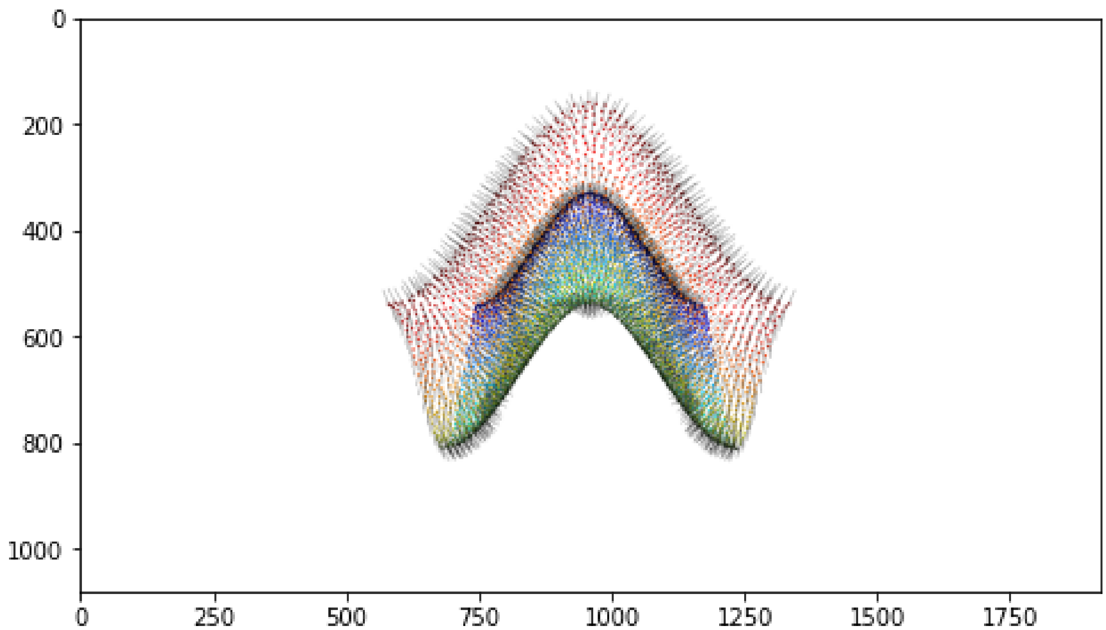
[35]:
```
from sklearn.decomposition import PCA
from sklearn.neighbors import KDTree
```

[36]:
```
tree = KDTree(new_points)
```

[37]:
```
normals = []
for p in new_points:
    neighbors = tree.query([p], 50)[1][0]
    neighbors = new_points[neighbors]
    normal = np.linalg.svd(neighbors - p)[2][2]
    normals.append(normal)
normals = np.array(normals)
# normals *= np.sign(normals[:, [1]])
```

```
direction = new_points - (new_points.min(0) + new_points.max(0)) / 2
normals = np.sign(np.sum(normals * direction, -1))[:, None] * normals
```

[38]:
```
import open3d
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(new_points)
pcd.normals = open3d.utility.Vector3dVector(normals)
draw_geometries([pcd])
```




4.4 your solution here

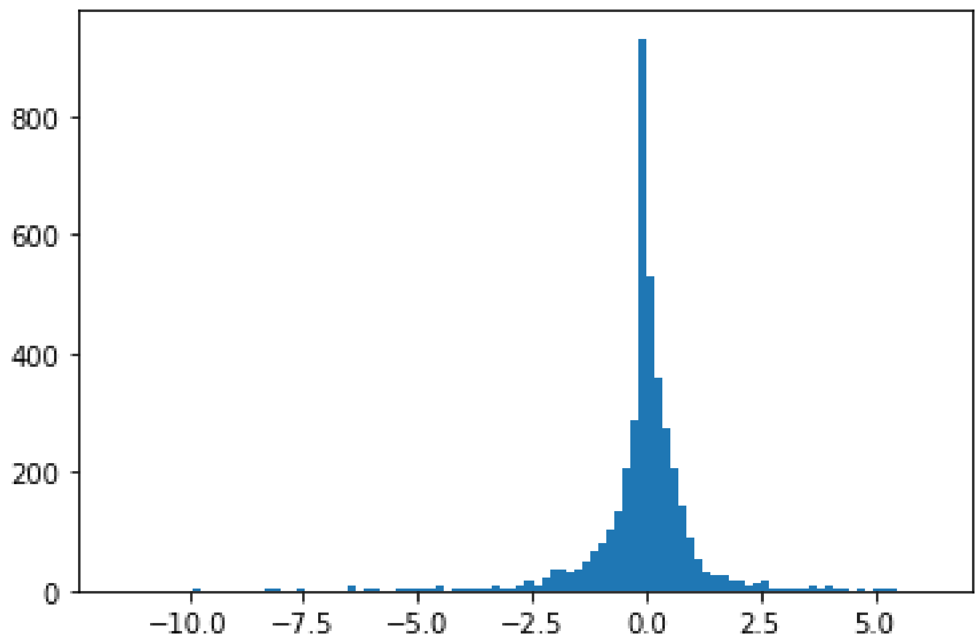
[39]:
```
k1s = []
k2s = []
for i,(point,normal) in enumerate(zip(new_points, normals)):
    idx = [idx for idx in tree.query([point], 20)[1][0] if idx != i]
    neighbors = new_points[idx]
    neighbor_normals = normals[idx]

    npq_normal = np.array([np.cross(normal, n - point) for n in neighbors])
    npq_normal = np.array([n/np.linalg.norm(n) for n in npq_normal])

    sin_beta = np.array([np.cross(normal, n)@n2 for n,n2 in␣
 ↪zip(neighbor_normals, npq_normal)])
    pq_sin_alpha = np.array([np.linalg.norm(np.cross(n - point, -normal)) for n␣
 ↪in neighbors])
```

```
    k = -sin_beta / pq_sin_alpha

    # build frame
    if abs(normal @ np.array([1,0,0])) > 0.95:
        X = np.cross(normal, np.array([0,1,0]))
    else:
        X = np.cross(normal, np.array([1,0,0]))
    X /= np.linalg.norm(X)
    Y = np.cross(normal, X)

    X_comp = np.array([X @ (n - point) for n in neighbors])
    Y_comp = np.array([Y @ (n - point) for n in neighbors])
    length = (X_comp ** 2 + Y_comp ** 2) ** 0.5
    cos_theta = X_comp / length
    sin_theta = Y_comp / length

    M = np.zeros((cos_theta.shape[0], 3))
    M[:, 0] = cos_theta ** 2
    M[:, 1] = 2 * cos_theta * sin_theta
    M[:, 2] = sin_theta ** 2

    A,B,C = np.linalg.lstsq(M, k, rcond=None)[0]
    k1, k2 = np.linalg.eigh([[A,B],[B,C]])[0]
    k1s.append(k1)
    k2s.append(k2)
k1s = np.array(k1s)
k2s = np.array(k2s)
```

[40]:
```
g = k1s * k2s
vis_g = g
plt.hist(g, bins=100)
```

[40]:
```
(array([  1.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   2.,   0.,
          0.,   1.,   1.,   0.,   0.,   0.,   1.,   3.,   3.,   0.,   1.,
          2.,   1.,   0.,   1.,   1.,   0.,   6.,   0.,   2.,   2.,   1.,
          1.,   3.,   2.,   2.,   4.,   5.,   7.,   1.,   3.,   3.,   4.,
          4.,   4.,   6.,   5.,   5.,  10.,  18.,   9.,  22.,  35.,  34.,
         31.,  36.,  47.,  67.,  78., 101., 136., 204., 286., 932., 528.,
        360., 274., 204., 141.,  90.,  51.,  32.,  28.,  25.,  17.,  18.,
         10.,  11.,  18.,   4.,   5.,   4.,   4.,   4.,   6.,   2.,   9.,
          4.,   4.,   1.,   3.,   1.,   2.,   2.,   2.,   0.,   0.,   0.,
          1.]),
 array([-11.53188429, -11.35423936, -11.17659444, -10.99894952,
        -10.8213046 , -10.64365967, -10.46601475, -10.28836983,
        -10.11072491,  -9.93307999,  -9.75543506,  -9.57779014,
         -9.40014522,  -9.2225003 ,  -9.04485537,  -8.86721045,
         -8.68956553,  -8.51192061,  -8.33427568,  -8.15663076,
```

```
        -7.97898584,  -7.80134092,  -7.62369599,  -7.44605107,
        -7.26840615,  -7.09076123,  -6.9131163 ,  -6.73547138,
        -6.55782646,  -6.38018154,  -6.20253662,  -6.02489169,
        -5.84724677,  -5.66960185,  -5.49195693,  -5.314312  ,
        -5.13666708,  -4.95902216,  -4.78137724,  -4.60373231,
        -4.42608739,  -4.24844247,  -4.07079755,  -3.89315262,
        -3.7155077 ,  -3.53786278,  -3.36021786,  -3.18257294,
        -3.00492801,  -2.82728309,  -2.64963817,  -2.47199325,
        -2.29434832,  -2.1167034 ,  -1.93905848,  -1.76141356,
        -1.58376863,  -1.40612371,  -1.22847879,  -1.05083387,
        -0.87318894,  -0.69554402,  -0.5178991 ,  -0.34025418,
        -0.16260926,   0.01503567,   0.19268059,   0.37032551,
         0.54797043,   0.72561536,   0.90326028,   1.0809052 ,
         1.25855012,   1.43619505,   1.61383997,   1.79148489,
         1.96912981,   2.14677474,   2.32441966,   2.50206458,
         2.6797095 ,   2.85735442,   3.03499935,   3.21264427,
         3.39028919,   3.56793411,   3.74557904,   3.92322396,
         4.10086888,   4.2785138 ,   4.45615873,   4.63380365,
         4.81144857,   4.98909349,   5.16673842,   5.34438334,
         5.52202826,   5.69967318,   5.87731811,   6.05496303,
         6.23260795]),
<BarContainer object of 100 artists>)
```




[41]:
```
colors = np.array([0,0,-1]) * vis_g[:,None].clip(-1, 0) + np.array([1,0,0]) *
↪vis_g[:,None].clip(0, 1)
```

```
[42]: import open3d
pcd = open3d.geometry.PointCloud()
pcd.points = open3d.utility.Vector3dVector(new_points)
pcd.colors = open3d.utility.Vector3dVector(colors)
draw_geometries([pcd])
```