Homework 1

1 QCQP [5pt]

Problem 1. In this exercise, you will design an efficient solver to a common quadratically-constrained optimization problem, and will use it to align 3D geometries in Problem 3.

Let's consider the following least square problem:

$$\begin{aligned} \text{minimize } & \frac{1}{2}\|Ax-b\|_2^2 \\ \text{subject to } & x^T x \le \epsilon \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $\epsilon \in \mathbb{R}$, $\epsilon > 0$, the variable is $x \in \mathbb{R}^n$. We assume $rank(A) = n$.

The solution to this problem will be used in Problem 3 and we need to design an effective solver. The solution to this optimization problem has to satisfy the Karush-Kuhn-Tuker (KKT) conditions:

$$\begin{cases} \nabla_x \mathcal{L}(x,\lambda) = 0 & (1) \\ x^T x \le \epsilon & (2) \\ \lambda \ge 0 & (3) \\ \lambda(x^T x - \epsilon) = 0 & (4) \end{cases}$$

where $\mathcal{L} = \frac{1}{2}\|Ax-b\|_2^2 + \lambda(x^T x - \epsilon)$ is the Lagrangian of the problem. We will use these conditions without proof.

1. Write down the gradient of the Lagrangian $\nabla_x \mathcal{L}$.

   Next, we will solve the problem by considering two cases based on Condition (2).

2. Case 1: $x^T x < \epsilon$. Then, $\lambda = 0$ by Condition (4). To satisfy Condition (1), it is equivalent to solving the unconstrained least square problem with objective $\frac{1}{2}\|Ax-b\|_2^2$. If its solution satisfies Condition (2), we are done. Write down the closed-form solution to the unconstrained least-square problem.

3. Case 2: $x^T x = \epsilon$.

   (a) Set $\nabla_x \mathcal{L} = 0$, express x in terms of λ, i.e. $x = h(\lambda)$.

   (b) Prove that $h(\lambda)^T h(\lambda)$ is monotonically decreasing for $\lambda \geq 0$. Hint: You might need the fact that $A^T A = U \Lambda U^T$.

   By the monotone property of $h(\lambda)^T h(\lambda)$, we can solve $x^T x = \epsilon$ by line search over λ (e.g. bisection method or Newton's iterative method). You will implement it in the following programming assignment.

4. (Programming assignment) Solve a provided instance of this problem.

   - You can load the data **QCQP.npz** by:

```
import numpy as np
data = np.load("./QCQP.npz")
A = data["A"]
b = data["b"]
eps = data["eps"]
```

   - You are not allowed to use external optimization libraries that solve this problem directly. Linear algebra functions in *numpy* are allowed (e.g. numpy.linalg.inv, numpy.linalg.eig, numpy.linalg.svd, numpy.linalg.lstsq, etc.).

2 3D Geometry Processing [8+2pt]

Problem 2. In this exercise, you will practice common processing routines of point cloud and 3D mesh. All questions are programming assignments.

1. Given mesh **saddle.obj**, sample *100K* points uniformly on the surface. You may use a library(such as *trimesh* or *open3d*) to do it.

2. Use iterative farthest point sampling method to sample *4K* points from the *100K* uniform samples. You need to implement this algorithm. You may only use computation libraries such as *Numpy* and *Scipy*. Hint: To accelerate computation, use Numpy matrix operations whenever possible.

3. Normal estimation: At each point of the *4K* points, estimate the normal vector by Principal Component Analysis using *50* nearest neighbors from the *100K* uniform points(You can use *sklearn.decomposition.PCA*). Since making the direction of normals consistent is a non-trivial task, for this assignment, you can orient the normals so that they roughly points in the Y direction.

   Note 1: You can use *sklearn.neighbors.KDTree* to efficiently extract nearest neighbors.

   Note 2: We also provide a noisy saddle point cloud **saddle.ply**. If you want, you can try some additional strategies (e.g. RANSAC) to improve the robustness of your method.

4. Mesh curvature estimation: Rusinkiewicz's method is a an effective method for face curvature estimation. Given a 3D triangular mesh, we assume the surface function can be parameterized as $f(u, v) \in \mathbb{R}^3$ in a local small triangle, where $[u, v] \in \mathbb{U}$ is a 2D coordinate. Since the triangle is small, we also assume that the tangent planes at each of the three vertices are approximately parallel, and $\mathbb{U}$ is roughly aligned with these tangent planes (i.e. $D_f \approx [\xi_u, \xi_v]$, where D_f denotes the first-order derivative of f). Since f can be arbitrarily defined, ξ_u and ξ_v can be defined as orthogonal vectors for simplicity.

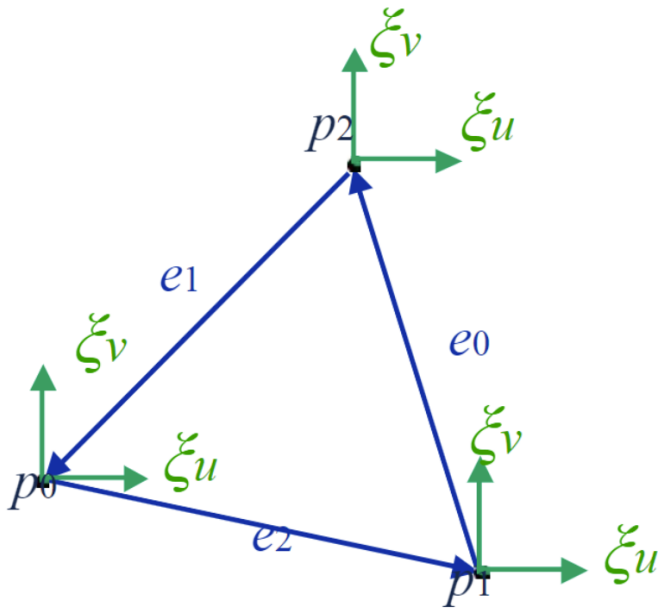

Figure 1: Assumptions in Rusinkiewicz's method.

Let $D_N = [\frac{\partial \vec{n}}{\partial u}, \frac{\partial \vec{n}}{\partial v}] \in \mathbb{R}^{3\times 2}$ denotes the first-order derivative of the normal. For each point on the surface, we define $S \in \mathbb{R}^{2\times 2}$ that satisfies $D_N = D_f \cdot S$. S describes the normal change when the point moves along any direction. The principle curvatures at the point can be defined as the two eigenvalues of S.

When we consider a small triangle on a mesh, $S = D_f^T D_N$ since $D_f = [\xi_u, \xi_v]$. According to the approximation that $D_N \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \approx \Delta \vec{n}$ and $D_f \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \approx \vec{e} = \Delta p$, we have

$$\begin{cases} SD_f^T(p_2 - p_1) = D_f^T(\vec{n_2} - \vec{n_1}), \\ SD_f^T(p_0 - p_2) = D_f^T(\vec{n_0} - \vec{n_2}), \\ SD_f^T(p_1 - p_0) = D_f^T(\vec{n_1} - \vec{n_0}), \end{cases} \tag{1}$$

where we use subscripts $\{0, 1, 2\}$ to distinguish three different vertices of the triangle. We can directly compute S by the least-square method (4 variables and 6 equations).

We can build up the following routine to implement Rusinkiewicz's method:

- Step 1: Load p_i, $\vec{n_i}$, and the face normal $\vec{n}$
- Step 2: Select two orthogonal vectors ξ_u and ξ_v on the tangent plane of the face
- Step 3: Define $D_f = [\xi_u, \xi_v]$, design the optimization problem by Equation 1 and compute S by least-square
- Step 4: Compute the eigenvalues of S as the principal curvatures

Use Rusinkiewicz's method to compute the principle curvatures for each triangular face in **icosphere.obj** and **sievert.obj**. Show your results by histograms. You may use *trimesh* or *open3d* to load the mesh and compute normals. However, you should not use any function that computes curvatures (the function may not compute them correctly).

In fact, the surface (**Sievert's surface**) in **sievert.obj** is locally isometric to a region on a sphere! If you compute Gaussian curvature (the product of principle curvatures), then you can find the Gaussian curvatures of most points equal to a constant 1.

5. [extra credit] Curvature estimation for point cloud: Modify the Rusinkiewicz's method used on mesh and apply it to the point cloud that you sampled in Question 2. Describe your algorithm and plot the Gaussian curvature for the shape.

3 Rotation [7+1pt]

Problem 3. Let $p = (1 + i)/\sqrt{2}$ and $q = (1 + j)/\sqrt{2}$, denote the unit-norm quaternions. The rotation $M(p)$ is a 90-degree rotation about the X axis, while $M(q)$ is a 90-degree rotation about the Y axis. Here we composed the two rotations $M(p)$ and $M(q)$. Here, we instead investigate the rotation that lies halfway between $M(p)$ and $M(q)$.

The quaternion that lies halfway between p and q is simply

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

1. Calculate the norm $|(p+q)/2|$ of that quaternion, and note that it is not 1. Find a quaternion r that is a scalar multiple of $(p+q)/2$ and that has unit norm, $|r| = 1$, and calculate the rotation matrix $M(r)$. Around what axis does $M(r)$ rotate, and through what angle(to the nearest tenth of a degree)?

2. What are the exponential coordinates of p and q?

3. Skew-symmetric representation of rotation: In this problem, we use $[\omega]$ to represent a skew-symmetric matrix constructed from $\omega \in \mathbb{R}^3$ as instructed in class:

   (a) Build the skew-symmetric matrix $[\omega_p]$ of p and $[\omega_q]$ of q, and derive their rotation matrices.

   (b) Using what you have above to verify that the following $\exp([\omega_1] + [\omega_2]) = \exp([\omega_1])\exp([\omega_2])$ relationship does **not** hold for exponential map in general(Note: The condition for this equation to hold is $[\omega_1][\omega_2] = [\omega_2][\omega_1]$). Therefore, composing rotations in skew-symmetric representation should not be done in this way.

   (c) Given two point clouds $X, Y \in \mathbb{R}^{3\times n}$ (**teapots.npz**) sampled from the surfaces of two shapes, we aim to estimate the rigid transformation to best align them. For simplicity, assume that we have found the correspondence between the two point clouds, so that the j-th column of X and Y are matched points. We also assume that the two point clouds are already aligned by translation. Then, the point cloud alignment problem can be formulated as the following Stiefel manifold optimization problem:

$$\begin{cases} \underset{R}{\text{minimize}} & \|RX - Y\|_F^2, \\ \text{subject to} & R^T R = I, \\ & det(R) = I. \end{cases}$$

   We can solve it using our knowledge of the exponential map. Given a rotation matrix R_1, rotation matrices in its local neighborhood R_2 can be parameterized as $R_2 = R_1 \exp([\Delta\omega]) \approx R_1(I + [\Delta\omega])$ for $\Delta\omega \approx 0$.

   (i) Use the above knowledge to build an optimization algorithm by filling in the following routine:

   - Step 1: Initiate $R := I$
   - Step 2: Describe your process to find a Δw

     Note: Convert the objective to a linear least square problem and solve it by your solver in Problem 1:

$$\begin{cases} \underset{\Delta\omega}{\text{minimize}} & \|A\Delta w - B\|_F^2 \\ \text{subject to} & \|\Delta\omega\|^2 \leq \epsilon \end{cases}$$

     Hint: $[v_1]v_2 = v_1 \times v_2 = -[v_2]v_1$

- Step 3: Update R by $R := R\exp[\Delta w]$
- Step 4: Go to step 2

(ii) (Programming assignment) Use your algorithm to solve the point cloud data alignment problem with the provided data **teapots.npz**.

You can load the data **teapots.npz** by:

```
import numpy as np
data = np.load("./teapots.npz")
X = data["X"]
Y = data["Y"]
```

Note: For the point cloud alignment problem under rigid transformation, there exists a closed-form solution to be covered in the second half semester of our course. While the closed-form solution is more efficient, this iterative solver has better flexibility and is often used for solving non-rigid alignment problems.

4. Double-covering of quaternion: What are the exponential coordinates of $p' = -p$ and $q' = -q$? What do you observe by comparing the exponential coordinates of $(p, -p)$ and $(q, -q)$? Does this relation hold for any quaternion pair $(r, -r)$? If it does, write down the statement and prove it.

5. [extra credit] Consider what you have discovered in Question 4. When we design a neural network to regress $SO(3)$ represented as quaternion, is it proper to use L2-loss measuring the difference between the ground-truth and the prediction? What if we regress a rotation matrix instead of quaternion?