Assignment

3D Reconstruction[1]

Make sure to start early!

Part I: Theory

Before implementing our own 3D reconstruction, let's take a look at some simple theory questions that may arise. The answers to the below questions should be relatively short, consisting of a few lines of math and text (maybe a diagram if it helps your understanding).

Q1.1 (5 points) Suppose two cameras fixate on a point P (see Figure 1) in space such that their principal axes intersect at that point. Show that if the image coordinates are normalized so that the coordinate origin $(0, 0)$ coincides with the principal point, the $\mathbf{F}_{33}$ element of the fundamental matrix is zero.

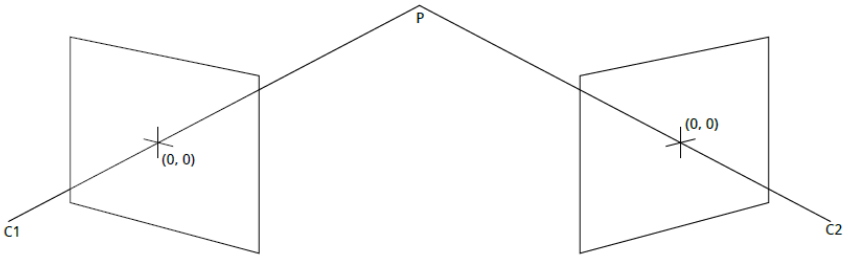

Figure 1: Figure for **Q1.1**. $C1$ and $C2$ are the optical centers. The principal axes intersect at point P.

Q1.2 (5 points) Consider the case of two cameras viewing an object such that the second camera differs from the first by a pure translation that is parallel to the x-axis. Show that the epipolar lines in the two cameras are also parallel to the x-axis. Backup your argument with relevant equations.

Q1.3 (5 points) Suppose we have an inertial sensor which gives us the accurate positions (R_i and t_i, where R is the rotation matrix and t is corresponding translation vector) of the robot at time i. What will be the effective rotation (R_{rel}) and translation (t_{rel}) between two frames at different time stamps? Suppose the camera intrinsics (K) are known, express the essential matrix ($\mathbf{E}$) and the fundamental matrix ($\mathbf{F}$) in terms of K, R_{rel} and t_{rel}.

Q1.4 (10 points) Suppose that a camera views an object and its reflection in a plane mirror.

Show that this situation is equivalent to having two images of the object which are related by a skew-symmetric fundamental matrix. You may assume that the object is flat, meaning that all points on the object are of equal distance to the mirror (*Hint*: draw the relevant vectors to understand the relationships between the camera, the object and its reflected image.)

[1]Credit to CMU Simon Lucey, Chengqian Che, Nate Chodosh, Allie Chang, Akshita Mittel, Gaurav Mittal, Sowmya Munukutla

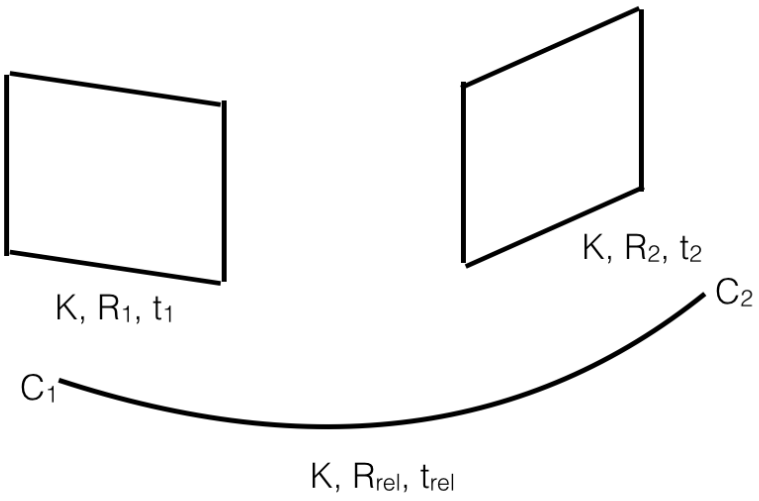

Figure 2: Figure for **Q1.3**. $C1$ and $C2$ are the optical centers. The rotation and the translation is obtained using inertial sensors. R_{rel} and t_{rel} are the relative rotation and translation between two frames.

Part II: Practice

1 Overview

In this part you will begin by implementing the two different methods seen in class to estimate the fundamental matrix from corresponding points in two images (Section 2). Next, given the fundamental matrix and calibrated intrinsics (which will be provided) you will compute the essential matrix and use this to compute a 3D metric reconstruction from 2D correspondences using triangulation (Section 3). Then, you will implement a method to automatically match points taking advantage of epipolar constraints and make a 3D visualization of the results (Section 4). Finally, you will implement RANSAC and bundle adjustment to further improve your algorithm (Section 5).

2 Fundamental matrix estimation

In this section you will explore different methods of estimating the fundamental matrix given a pair of images. In the `data/` directory, you will find two images (see Figure 3) from the Middlebury multiview dataset[2], which is used to evaluate the performance of modern 3D reconstruction algorithms.

The Eight Point Algorithm

The 8-point algorithm (discussed in class, and outlined in Section 10.1 of Forsyth & Ponce) is arguably the simplest method for estimating the fundamental matrix. For this section, you can use

[2] `http://vision.middlebury.edu/mview/data/`

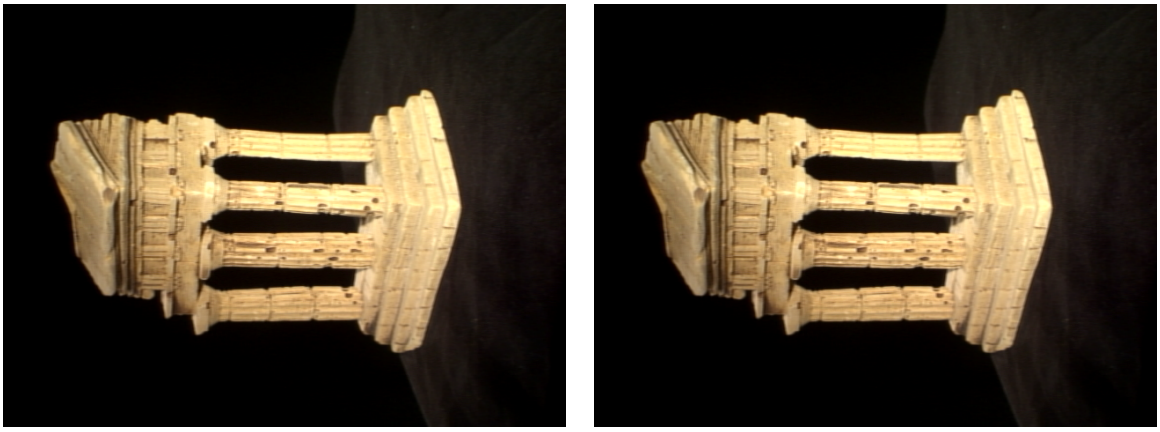

Figure 3: Temple images for this assignment

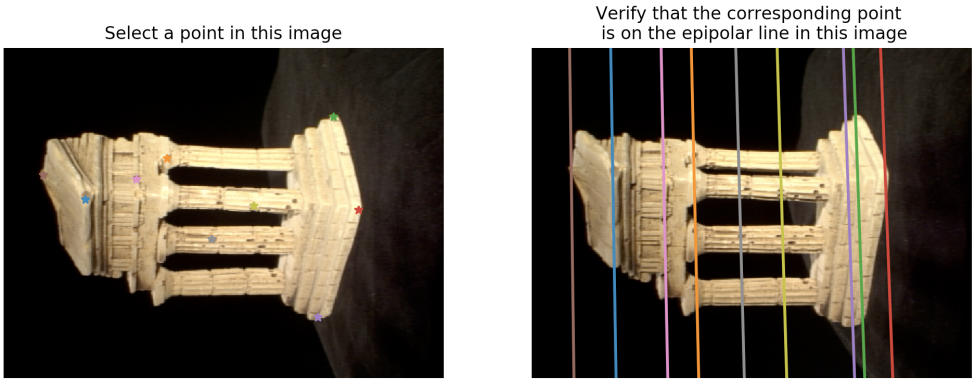

Figure 4: `displayEpipolarF` in `helper.py` creates a GUI for visualizing epipolar lines

provided correspondences you can find in `data/some_corresp.npz`.

Q2.1 (10 points) Finish the function eightpoint in `submission.py`. Make sure you follow the signature for this portion of the assignment:

```
F = eightpoint(pts1, pts2, M)
```

where `pts1` and `pts2` are $N \times 2$ matrices corresponding to the (x, y) coordinates of the N points in the first and second image repectively. M is a scale parameter.

- You should scale the data as was discussed in class, by dividing each coordinate by M (the maximum of the images width and height). After computing $\mathbf{F}$, you will have to "unscale" the fundamental matrix.

  Hint: If $x_{\text{normalized}} = Tx$, then $F_{\text{unnormalized}} = T^T \mathbf{F} T$.

  You must enforce the singularity condition of the $\mathbf{F}$ before unscaling.

- You may find it helpful to refine the solution by using local minimization. This probably wont fix a completely broken solution, but may make a good solution better by locally minimizing a geometric cost function.

For this we have provided a helper function `refineF` in `helper.py` taking in `F` and the two sets of points, which you can call from `eightpoint` before unscaling `F`.

- Remember that the x-coordinate of a point in the image is its column entry, and y-coordinate is the row entry. Also note that eight-point is just a figurative name, it just means that you need at least 8 points; your algorithm should use an over-determined system ($N > 8$ points).
- To visualize the correctness of your estimated `F`, use the supplied function `displayEpipolarF` in `helper.py`, which takes in `F`, and the two images. This GUI lets you select a point in one of the images and visualize the corresponding epipolar line in the other image (Figure 4).
- **Output**: Save your matrix `F`, scale `M` to the file `q2_1.npz`.

  In your write-up: Write your recovered `F` and include an image of some example output of `displayEpipolarF`.

The Seven Point Algorithm

Q2.2 (15 points) Since the fundamental matrix only has seven degrees of freedom, it is possible to calculate $\mathbf{F}$ using only seven point correspondences. This requires solving a polynomial equation. In the section, you will implement the seven-point algorithm (described in class, and outlined in Section 15.6 of Forsyth and Ponce). Manually select 7 points from provided point in data/some corresp.npz, and use these points to recover a fundamental matrix $\mathbf{F}$. The function should have the signature:

```
Farray = sevenpoint(pts1, pts2, M)
```

where `pts1` and `pts2` are 7×2 matrices containing the correspondences and `M` is the normalizer (use the maximum of the images height and width), and `Farray` is a list array of length either 1 or 3 containing Fundamental matrix/matrices. Use `M` to normalize the point values between $[0, 1]$ and remember to "unnormalize" your computed $\mathbf{F}$ afterwards.

- Use `displayEpipolarF` to visualize $\mathbf{F}$ and pick the correct one.
- **Output**: Save your matrix `F`, scale `M`, 2D points `pts1` and `pts2` to the file `q2_2.npz`.

  In your write-up: Write your recovered $\mathbf{F}$ and print an output of `displayEpipolarF`. Also, include an image of some example output of `displayEpipolarF` using the seven point algorithm.
- *Hints*: You can use Numpy's function `roots()`. The epipolar lines may not match exactly due to imperfectly selected correspondences, and the algorithm is sensitive to small changes in the point correspondences. You may want to try with different sets of matches.

3 Metric Reconstruction

You will compute the camera matrices and triangulate the 2D points to obtain the 3D scene structure. To obtain the Euclidean scene structure, first convert the fundamental matrix $\mathbf{F}$ to an essential matrix $\mathbf{E}$. Examine the lecture notes and the textbook to find out how to do this when the internal camera calibration matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ are known; these are provided in `data/intrinsics.npz`.

Q3.1 (5 points) Write a function to compute the essential matrix $\mathbf{E}$ given $\mathbf{F}, \mathbf{K}_1$ and $\mathbf{K}_2$ with the signature

```
E = essentialMatrix(F, K1, K2)
```

In your write-up: Write your estimated $\mathbf{E}$ using $\mathbf{F}$ from the eight-point algorithm. Given an essential matrix, it is possible to retrieve the projective camera matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ from it. Assuming $\mathbf{M}_1$ is fixed at $[\mathbf{I}, 0]$, $\mathbf{M}_2$ can be retrieved up to a scale and four-fold rotation ambiguity. For details on recovering $\mathbf{M}_2$, see section 7.2 in Szeliski. We have provided you with the function `camera2` in `python/helper.py` to recover the four possible $\mathbf{M}_2$ matrices given $\mathbf{E}$.

Note: The $\mathbf{M}_1$ and $\mathbf{M}_2$ here are projection matrices of the form: $M_1 = [I|0]$ and $M_2 = [R|t]$.

Q3.2 (10 points) Using the above, write a function to triangulate a set of 2D coordinates in the image to a set of 3D points with the signature:

```
[P, err] = triangulate(C1, pts1, C2, pts2)
```

where `pts1` and `pts2` are the $N \times 2$ matrices with the 2D image coordinates and `P` is an $N \times 3$ matrix with the corresponding 3D points per row. `C1` and `C2` are the 3×4 camera matrices. Remember that you will need to multiply the given intrinsics matrices with your solution for the canonical camera matrices to obtain the final camera matrices. Various methods exist for triangulation probably the most familiar for you is based on least squares (see Szeliski Chapter 7 if you want to learn about other methods).

For each point i, we want to solve for 3D coordinates $P_i = [x_i, y_i, z_i]^T$, such that when they are projected back to the two images, they are close to the original 2D points. To project the 3D coordinates back to 2D images, we first write P_i in homogeneous coordinates, and compute $\mathbf{C}_1 P_i$ and $\mathbf{C}_2 P_i$ to obtain the 2D homogeneous coordinates projected to camera 1 and camera 2, respectively.

For each point i, we can write this problem in the following form:

$$\mathbf{A}_i Pi = 0,$$

where $\mathbf{A}_i$ is a 4×4 matrix, and P_i is a 4×1 vector of the 3D coordinates in the homogeneous form. Then, you can obtain the homogeneous least-squares solution (discussed in class) to solve for each P_i.

In your write-up: Write down the expression for the matrix $\mathbf{A}_i$.

Once you have implemented triangulation, check the performance by looking at the reprojection error:

$$\texttt{err} = \sum_i \|p_{1i}, \hat{p}_{1i}\|^2 + \|p_{2i}, \hat{p}_{2i}\|^2$$

where $\hat{p}_{1i} = \text{Proj}(\mathbf{C}_1, P_i)$ and $\hat{p}_{2i} = \text{Proj}(\mathbf{C}_2, P_i)$.

Note: `C1` and `C2` here are projection matrices of the form: $\mathbf{C}_1 = \mathbf{K}_1\mathbf{M}_1 = \mathbf{K}_1[I|0]$ and $\mathbf{C}_2 = \mathbf{K}_2\mathbf{M}_2 = \mathbf{K}_2[R|t]$.

Q3.3 (10 points) Write a script `findM2.py` to obtain the correct `M2` from `M2s` by testing the four solutions through triangulations. Use the correspondences from `data/some_corresp.npz`.
Output: Save the correct `M2`, the corresponding `C2`, and 3D points `P` to `q3_3.npz`.

4 3D Visualization

You will now create a 3D visualization of the temple images. By treating our two images as a stereo-pair, we can triangulate corresponding points in each image, and render their 3D locations.

Q4.1 (15 points) Implement a function with the signature:

```
[x2, y2] = epipolarCorrespondence(im1, im2, F, x1, y1)
```

This function takes in the x and y coordinates of a pixel on `im1` and your fundamental matrix $\mathbf{F}$, and returns the coordinates of the pixel on `im2` which correspond to the input point. The match is obtained by computing the similarity of a small window around the (x_1, y_1) coordinates in `im1` to various windows around possible matches in the `im2` and returning the closest.

Instead of searching for the matching point at every possible location in `im2`, we can use $\mathbf{F}$ and simply search over the set of pixels that lie along the epipolar line (recall that the epipolar line passes through a single point in `im2` which corresponds to the point (x_1, y_1) in `im1`.

There are various possible ways to compute the window similarity. For this assignment, simple methods such as the Euclidean or Manhattan distances between the intensity of the pixels should suffice. See Szeliski Chapter 11, on stereo matching, for a brief overview of these and other methods. *Implementation hints*:

- Experiment with various window sizes.
- It may help to use a Gaussian weighting of the window, so that the center has greater influence than the periphery.
- Since the two images only differ by a small amount, it might be beneficial to consider matches for which the distance from (x_1, y_1) to (x_2, y_2) is small.

To test your `epipolarCorrespondence`, we have included a helper function `epipolarMatchGUI` in `python/helper.py`, which takes in two images the fundamental matrix. This GUI allows you to click on a point in `im1`, and will use your function to display the corresponding point in `im2`. See Figure 5.

It is not necessary for your matcher to get *every* possible point right, but it should get easy points (such as those with distinctive, corner-like windows). It should also be good enough to render an intelligible representation in the next question.

Output: Save the matrix `F`, points `pts1` and `pts2` which you used to generate the screenshot to the file `q4_1.npz`.

In your write-up: Include a screenshot of `epipolarMatchGUI` with some detected correspondences.

Q4.2 (10 points) Included in this homework is a file `data/templeCoords.npz` which contains 288 hand-selected points from `im1` saved in the variables `x1` and `y1`.

Now, we can determine the 3D location of these point correspondences using the triangulate function. These 3D point locations can then plotted using the Matplotlib or plotly package. Write a script `visualize.py`, which loads the necessary files from `../data/` to generate the 3D reconstruction using scatter. An example is shown in Figure 6.

Output: Again, save the matrix `F`, matrices `M1`, `M2`, `C1`, `C2` which you used to generate the screenshots to the file `q4_2.npz`.

In your write-up: Take a few screenshots of the 3D visualization so that the outline of the temple is clearly visible, and include them with your homework submission.

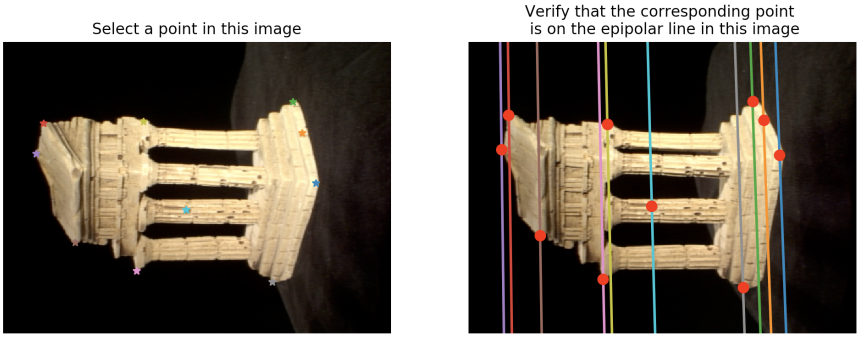

Figure 5: `epipolarMatchGUI` shows the corresponding point found by calling `epipolarCorrespondence`

5 Bundle Adjustment

Q5.1 (15 points) In some real world applications, manually determining correspondences is infeasible and often there will be noisy coorespondances. Fortunately, the RANSAC method seen in class can be applied to the problem of fundamental matrix estimation.

Implement the above algorithm with the signature:

```
[F, inliers] = ransacF(pts1, pts2, M)
```

where $\mathbf{M}$ is defined in the same way as in Section 2 and inliers is a boolean vector of size equivalent to the number of points. Here inliers is set to true only for the points that satisfy the threshold defined for the given fundamental matrix $\mathbf{F}$.

We have provided some noisy coorespondances in some `corresp_noisy.npz` in which around 75% of the points are inliers. Compare the result of RANSAC with the result of the eightpoint when ran on the noisy coorespondances. Briefly explain the error metrics you used, how you decided which points were inliers and any other optimizations you may have made.

Hints: Use the seven point to compute the fundamental matrix from the minimal set of points. Then compute the inliers, and refine your estimate using all the inliers.

Q5.2 (15 points)

So far we have independently solved for camera matrix, $\mathbf{M}_j$ and 3D projections, P_i. In bundle adjustment, we will jointly optimize the reprojection error with respect to the points Pi and the camera matrix $\mathbf{M}_j$.

$$\texttt{err} = \sum_{ij} \|p_{ij} - \text{Proj}(\mathbf{C}_j, P_i)\|^2$$

where $\mathbf{C}_j = \mathbf{K}_j\mathbf{M}_j$, same as in **Q3.2**.

For this homework we are going to only look at optimizing the extrinsic matrix. To do this we will be parametrizing the rotation matrix $\mathbf{R}$ using Rodrigues formula to produce vector $\mathbf{r} \in \mathbb{R}^3$. Write a function that converts a Rodrigues vector $\mathbf{r}$ to a rotation matrix $\mathbf{R}$

```
R = rodrigues(r)
```

as well as the inverse function that converts a rotation matrix $\mathbf{R}$ to a Rodrigues vector $\mathbf{r}$

```
r = invRodrigues(R)
```

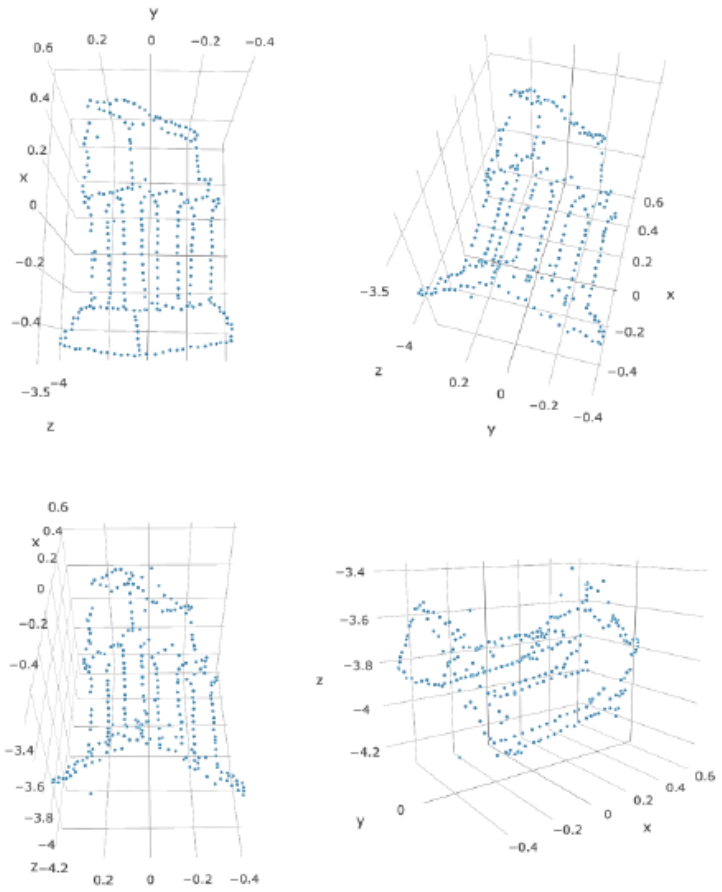

Figure 6: An example point cloud

Q5.3 (10 points)

Using this parameterization, write an optimization function

```
residuals = rodriguesResidual(K1, M1, p1, K2, p2, x)
```

where `x` is the flattened concatenation of $\mathbf{P}$, $\mathbf{r}_2$, and $\mathbf{t}_2$. $\mathbf{P}$ are the 3D points; $\mathbf{r}_2$ and $\mathbf{t}_2$ are the rotation (in the Rodrigues vector form) and translation vectors associated with the projection matrix $\mathbf{M}_2$. The `residuals` are the difference between original image projections and estimated projections (the square of 2-norm of this vector corresponds to the error we computed in **Q3.2**):

```
residuals = numpy.concatenate([(p1-p1 hat).reshape([-1]),
               (p2-p2 hat).reshape([-1])])
```

Use this error function and Scipys nonlinear least square optimizer `leastsq` write a function to optimize for the best extrinsic matrix and 3D points using the inlier correspondences from some `corresp_noisy.npz` and the RANSAC estimate of the extrinsics and 3D points as an initialization.

```
[M2, P] = bundleAdjustment(K1, M1, p1, K2, M2 init, p2, P init)
```

In your write-up: include an image of the original 3D points and the optimized points as well as the reprojection error with your initial $\mathbf{M}_2$ and $\mathbf{P}$, and with the optimized matrices.

6 Deliverables

If your andrew id is `<ustid-login>`, your submission should be the writeup `<ustid-login>.pdf` contained in a zip file `<ustid-login>.zip`. Please submit the zip file to CASS.

The zip file should include the following directory structure:

- `submission.py`: your implementation of algorithms.
- `findM2.py`: script to compute the correct camera matrix.
- `visualize.py`: script to visualize the 3d points.
- `q2_1.npz`: file with output of Q2.1.
- `q2_2.npz`: file with output of Q2.2.
- `q3_3.npz`: file with output of Q3.3.
- `q4_1.npz`: file with output of Q4.1.
- `q4_2.npz`: file with output of Q4.2.