Before you begin

- Create a new file `inertia.py`, and download `eggholder.py` from Canvas.
- Refresh your memory on what the moment of inertia of a rigid body is. Here's a brief overview of the concept from `Khan Academy`.

Learning Objectives:

You should be able to do the following:

- Use Numpy to solve problems with mechanical significance
- Use scipy's utilities to perform minimization

Thoughts, come back after you have read the problems:

- Unlike the previous lab on Numpy, we're not explicitly preventing you from using loops for this assignment (in fact, one part of the assignment is more easily written using loops). However, you should still try to use vectorized operations wherever possible, because they take less lines of code, which in turn makes your code more readable, which counts for code style. In particular, Problem 1 can be done entirely without loops.

Grading Checkpoints [16 points]

Grading Checkpoints (12 points total + 3 extra credit) See the associated footnotes for information out rubric items * .

- [3 points] `compute_inertia_matrix()` produces the correct values * .
- [2 points] `sample_sphere_polar()` produces the expected distribution of values * .
- [2 points] `sample_sphere_gaussian()` produces the expected distribution of values * .
- [1 points] Plausible inertia matrices from random distributions (0.5 points) and a correct expected inertia matrix (0.5 points) are shown.
- [1 points] **Extra credit:** 3D plot shown with correct interpretation of discrepancy.
- [2 points] `minimize_eggholder()` returns the expected values (1 point) and is consistent with our implementation (1 point) * .
- [2 points] Your histogram exhibits the expected trends (1.5 points) with proper axes and title (0.5 points).

Standard Grading Checkpoints (4 points total)

- [2 points] Code passes PEP8 checks with 10 errors max * .
- [2 points] Code passes TA-reviewed style checks for cleanliness, layout, readability, etc.

Hand in all files to Gradescope.

*Indicates that the autograder will tell you if this problem is correct on our set of test cases and assign you credit.

Problem 1 Moment of Inertia Matrices

Put all code in a file called `inertia.py`. Recall the moment of inertia matrix, a symmetric 3×3 matrix which describes the tendency of a rigid body to resist motion. It is expressed in the following form:

$$\begin{bmatrix} \mathcal{I}_{xx} & \mathcal{I}_{xy} & \mathcal{I}_{xz} \\ \mathcal{I}_{xy} & \mathcal{I}_{yy} & \mathcal{I}_{yz} \\ \mathcal{I}_{xz} & \mathcal{I}_{yz} & \mathcal{I}_{zz} \end{bmatrix}$$

For a continuously-distributed body with a mass M and a density function described by $\rho(x, y, z)$, this inertia matrix would be computed by using an integral over the body of interest, which can be quite challenging to numerically solve. One way to get around this is by uniformly sampling points N from the body, each with a mass $m_i = \frac{M}{N}$, and then computing the inertia matrix as follows:

$$\mathcal{I}_b = \begin{bmatrix} \sum m_i(y_i^2 + z_i^2) & -\sum m_i x_i y_i & -\sum m_i x_i z_i \\ -\sum m_i x_i y_i & \sum m_i(x_i^2 + z_i^2) & -\sum m_i y_i z_i \\ -\sum m_i x_i z_i & -\sum m_i y_i z_i & \sum m_i(x_i^2 + y_i^2) \end{bmatrix}$$

In other words, we can get an approximation of the inertia matrix of the rigid body by treating it as the sum of the inertia matrices of point masses. We will apply this method to estimating the moment of inertia matrix for a hollow sphere with radius 1. For this, we need a way to sample points off the surface of a sphere. In this section, we will propose two ways of doing so and observe whether the computed inertia matrix matches what we would expect.

- Write a function `compute_inertia_matrix` which takes in one required argument, an $N \times 3$ array, as well as an optional parameter mass defaulting to 1 which represents the mass of the object being sampled from (`not` the mass of each point). It should return a 3×3 Numpy array representing the inertia matrix which uses the formula given above to compute the final matrix.

- One way to generate random points on a sphere surface is via the angle-azimuth representation, where the angle ϕ in $[0, \pi]$ represents how "elevated" the point is relative to the z-axis, and the azimuth θ in $[0, 2\pi]$ represents the angle of rotation in the xy-plane. We can then convert a pair (ϕ, θ) into a sphere point with the following formula (with r=1):

  $$\begin{aligned} x &= r\sin(\phi)\cos(\theta) \\ y &= r\sin(\phi)\sin(\theta) \\ z &= r\cos(\phi) \end{aligned}$$

  Write a function `sample_sphere_polar` which takes in a single value N, and returns an $N \times 3$ Numpy array with sphere points sampled by picking random pairs of (ϕ, θ) in the given ranges.

- Another way to generate random points on a sphere surface is to generate points with a Gaussian distribution, and rescale the points so that their magnitude is 1.
  Write a function `sample_sphere_gaussian` which takes in a single value N, and returns an $N \times 3$ Numpy array with sphere points sampled by generating an $N \times 3$ array where each value is drawn from a standard normal distribution (mean 0, stdev 1), and then rescaling each of the N points to have magnitude 1.

- For $n = 1000$ and a mass of 1, compute the inertia matrix by sampling points via `sample_sphere_polar` versus `sample_sphere_gaussian`, as well as `the expected inertia matrix` for $m = 1$ and $r = 1$. Write a function called `test_inertia_matrices_output()` which takes no arguments and prints the resulting inertia matrices as well as the expected inertia matrices exactly as

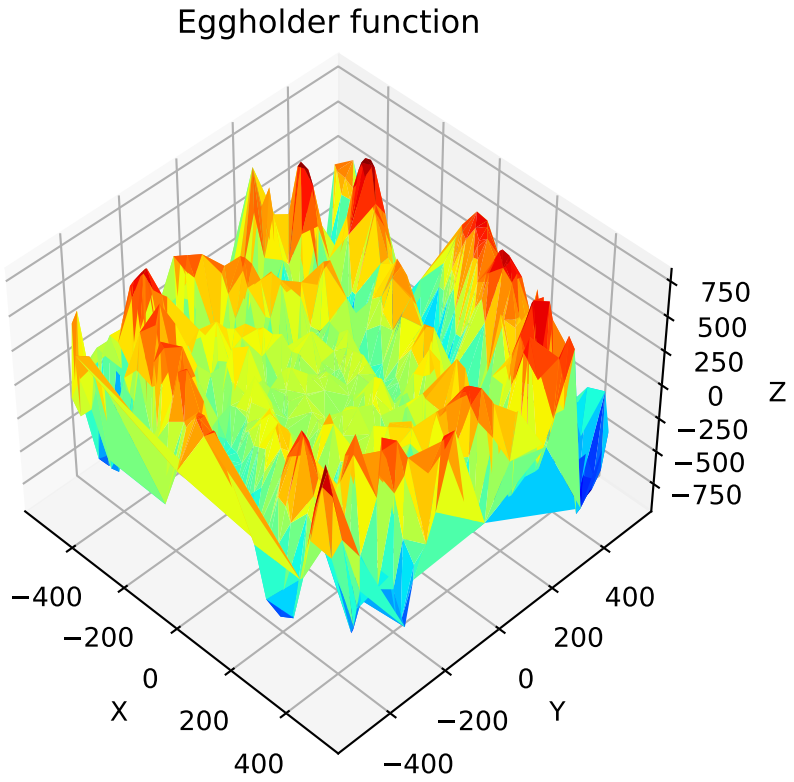
```
Polar:
[[a b c]
[a b c]
[a b c]]
Gaussian:
[[a b c]
[a b c]
[a b c]]
Expected:
[[a b c]
[a b c]
[a b c]]
```

where letters should be the actual value of items in your arrays. Use `np.set_printoptions(precision=3, suppress=True)` before printing to ensure the correct format when printing numpy arrays.

- **Extra credit:** If done correctly, one of these methods will show a deviation from the expectation. For this case, produce a `3D scatterplot` with Matplotlib showing the distribution of the points. Comment on the distribution of points and how they relate to the discrepancy. (What does it mean for something to have a low/high moment of inertia with respect to a given axis?)
- **Extra credit II:** There are other ways to generate evenly-placed points on the sphere, including: Dodecahedons, then subdividing, spherical sampling based on the way plants grow, using polar coordinate sampling for the middle part of the sphere – six times over, one for each "side" of the sphere, or modify the sample sphere gaussian to only take points that are within a thin shell around the sphere. Pick one (not necessarily one of these), describe it, and implement it as `sample_sphere_better`. Generate around 1,000 points.

Problem 2 Optimization

Download the file `eggholder.py`. This file contains a single function `eggholder` which takes in an x and y value, and returns a single value. A visualization of the function is shown here:



The goal here is to use fmin to see how well it can optimize this function. Put all of your code inside the same file `eggholder.py`.

- Write a function `minimize_eggholder(guess, max_calls=100)` which takes in an (x,y) guess as `guess`, and then uses the `fmin` function from scipy to attempt to minimize the function. It should set maxfun equal to `max_calls` to limit the number of function evaluations. It should return **two** values: The (x,y) coordinate which minimizes the function, followed by the actual value at the minimum.
- On the interval $[-512, 512] \times [-512, 512]$, the global minimum of the function is at $(512, 404.2319)$. Write some code which randomly generates 1000 points in the range $[-512, 512] \times [-512, 512]$ and runs `minimize_eggholder` with those points as an initial guess. It should then plot a histogram of the `absolute difference` between the minimum obtained

from `minimize_eggholder` versus the true global minimum. Set bins=25. Label your axes and include your plot in your submission.