|  |  |
| --- | --- |
| **Name:** | **Lab Time** F 0000 |

|  |
| --- |
| **Names of people you worked with:** |
|  |

|  |
| --- |
| **Websites you used:** |
|  |

|  |  |
| --- | --- |
| **Approximately how many hours did it take you to complete this assignment (to nearest whole number)?** |  |

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

**Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.**

**Turn .zip files of Python code to Canvas or your assignment will not be graded**

**BEFORE YOU BEGIN**

1. Create a new file inertia.py, and download eggholder.py from Canvas (Code/Labs/Lab 7).
2. 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](https://www.khanacademy.org/science/physics/torque-angular-momentum/torque-tutorial/a/rotational-inertia).

**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 and read these after you’ve read the problems):**

1. 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**

**Assignment Grading Checkpoints (12 points total + 3 extra credit)**

1. compute\_inertia\_matrix() produces the correct values. [3 points]\*
2. sample\_sphere\_polar() produces the expected distribution of values. [2 points]\*
3. sample\_sphere\_gaussian() produces the expected distribution of values. [2 points]\*
4. Plausible inertia matrices from random distributions (0.5 points) and a correct expected inertia matrix (0.5 points) are shown. [1 point total]
5. **Extra credit:** 3D plot shown with correct interpretation of discrepancy (1 point).
6. minimize\_eggholder() returns the expected values (1 point)\* and is consistent with our implementation (1 point)\*. [2 points total]
7. Your histogram exhibits the expected trends (1.5 points) with proper axes and title (0.5 points). [2 points total]

**Standard Grading Checkpoints (3.5 points total)**

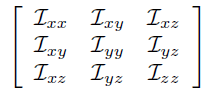
* [0.5 points] Turn in a properly filled-in PDF to Gradescope.
* [1 point]\* Code passes PEP8 checks with 10 errors max.
* [2 point] Code passes TA-reviewed style checks for cleanliness, layout, readability, etc.

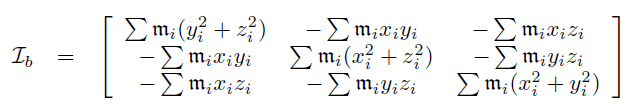
Hand in all files to Gradescope.

**Problem 1 Moment of Inertia Matrices**

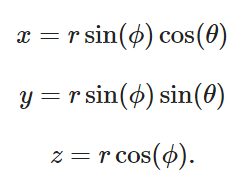
Put all code in a file called inertia.py.

Recall the moment of inertia matrix, a symmetric 3x3 matrix which describes the tendency of a rigid body to resist motion. It is expressed in the following form:



For a continuously-distributed body with a mass M and a density function described by ⍴(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 mi = M/N, and then computing the inertia matrix as follows:  
  


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.

1. Write a function compute\_inertia\_matrix which takes in one required argument, an N x 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 3x3 Numpy array representing the inertia matrix which uses the formula given above to compute the final matrix.
2. 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, 2pi] 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):  
     
     
     
   Write a function sample\_sphere\_polar which takes in a single value N, and returns an Nx3 Numpy array with sphere points sampled by picking random pairs of (ϕ, θ) in the given ranges.
3. 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 Nx3 Numpy array with sphere points sampled by generating an Nx3 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.
4. 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](https://en.wikipedia.org/wiki/List_of_moments_of_inertia) for m=1 and r=1. Fill in the values you get below (up to a reasonable number of decimal points, i.e. 2-3).
5. **Extra credit:** If done correctly, one of these methods will show a deviation from the expectation. For this case, produce a [3D scatterplot](https://matplotlib.org/3.1.1/gallery/mplot3d/scatter3d.html) 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?)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Using polar sampling | | | Using Gaussian sampling | | | Expected | | |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

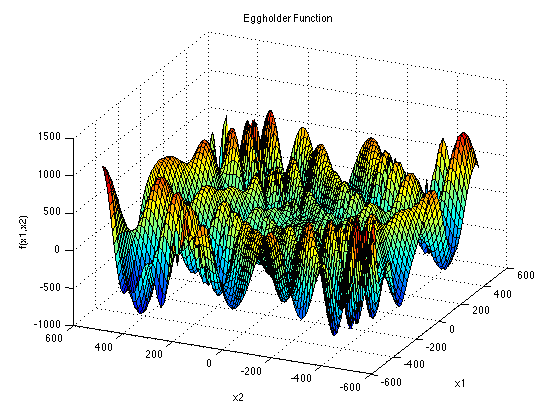
|  |
| --- |
| **Comments for grader/additional information (if any)** |
|  |

|  |
| --- |
| **Extra credit: 3D Plot** |
|  |

|  |
| --- |
| **Extra credit: Comment on the discrepancy.** |
|  |

**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.

1. Write a function minimize\_eggholder(guess, max\_calls=100) which takes in an (x,y) guess as guess, and then uses the [fmin](https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin.html) 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.
2. On the interval [-512, 512] x [-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] x [-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 paste the plot below.

|  |
| --- |
| **Comments for grader/additional information (if any)** |
|  |

|  |
| --- |
| **Eggholder optimization histogram** |
|  |