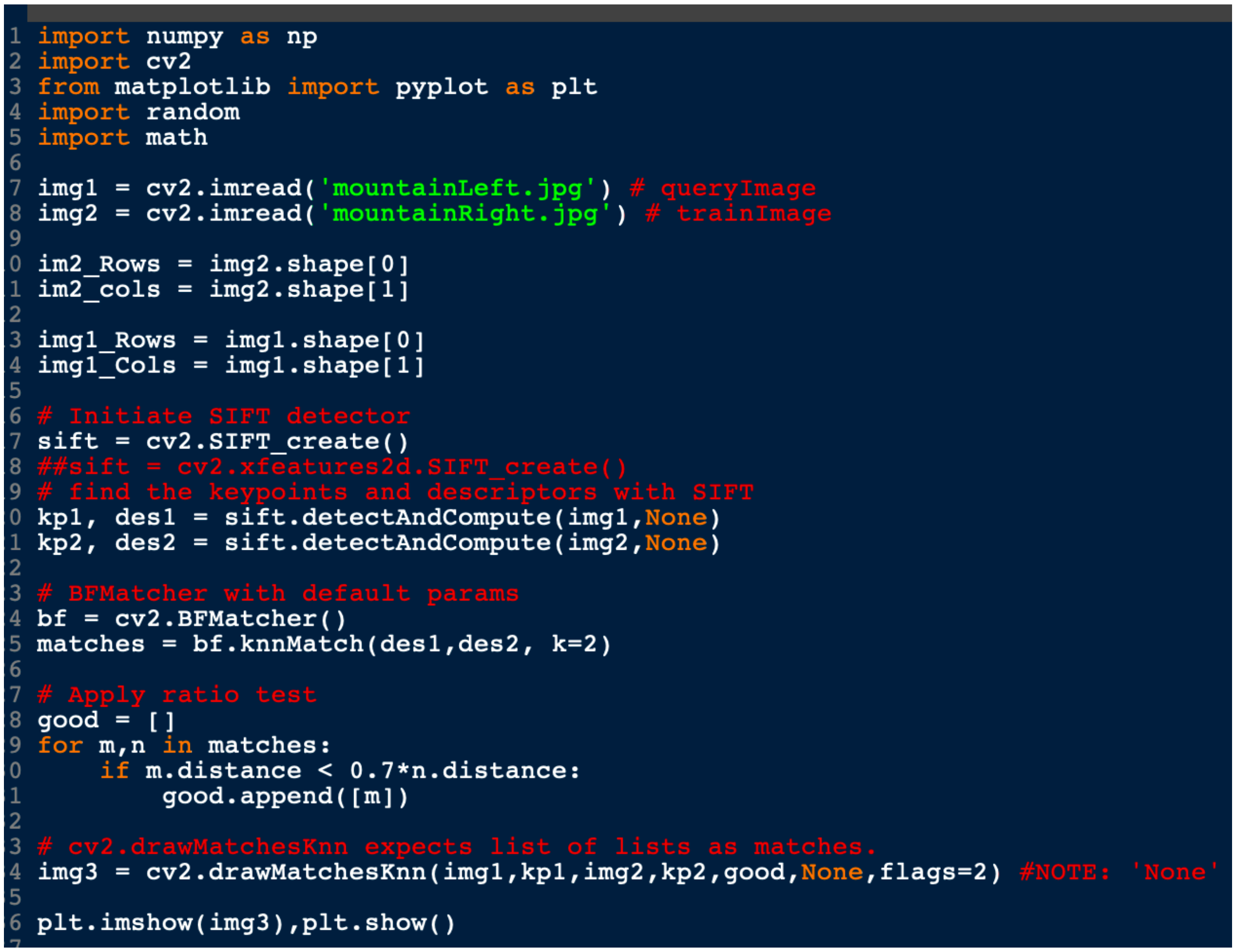
Image stitching part 1(Computing Best H)

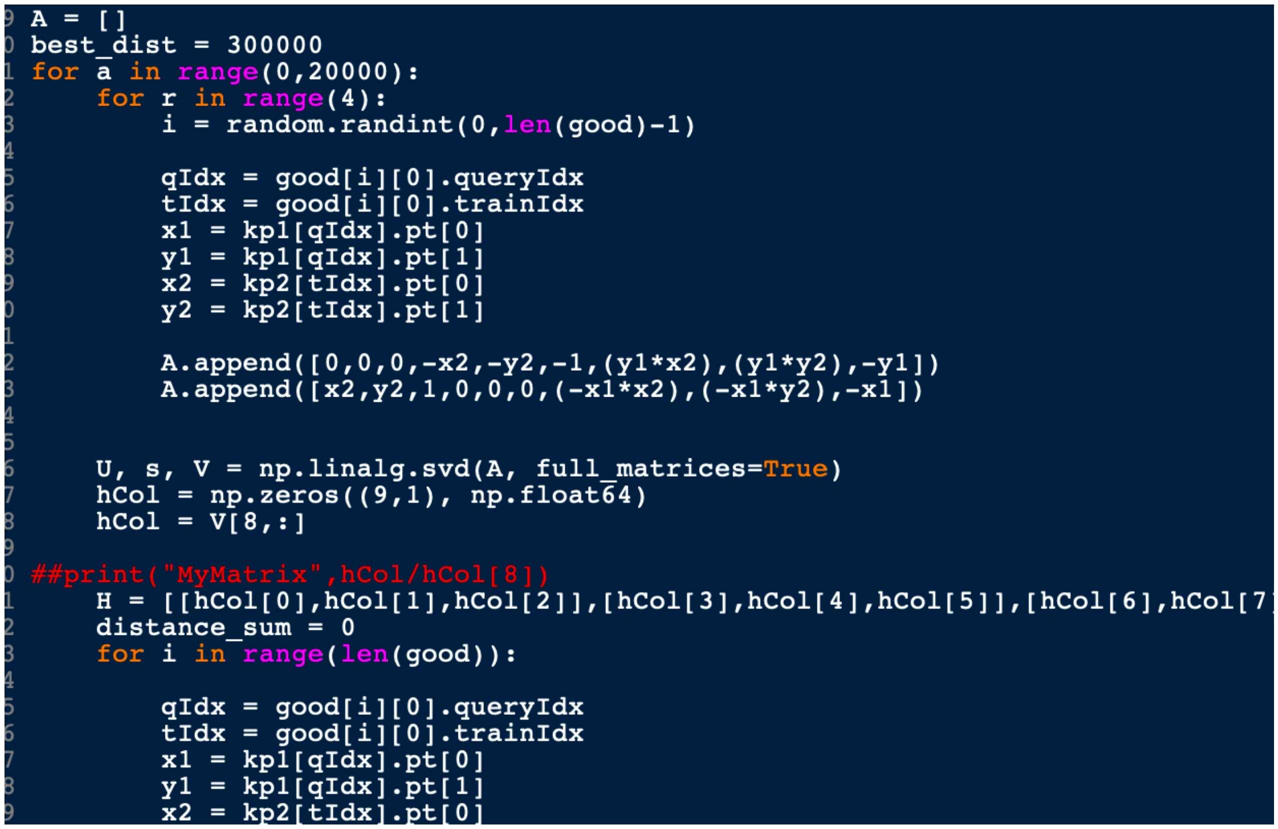
Kareem Tiwari

This first part of the image stitching assignment was for us to create a function that creates the best fitting Homography Matrix as compared to the one created by OpenCV. First, we are going to run SIFT to get the key point matches between the two images that are input in the program at the very beginning as shown below.



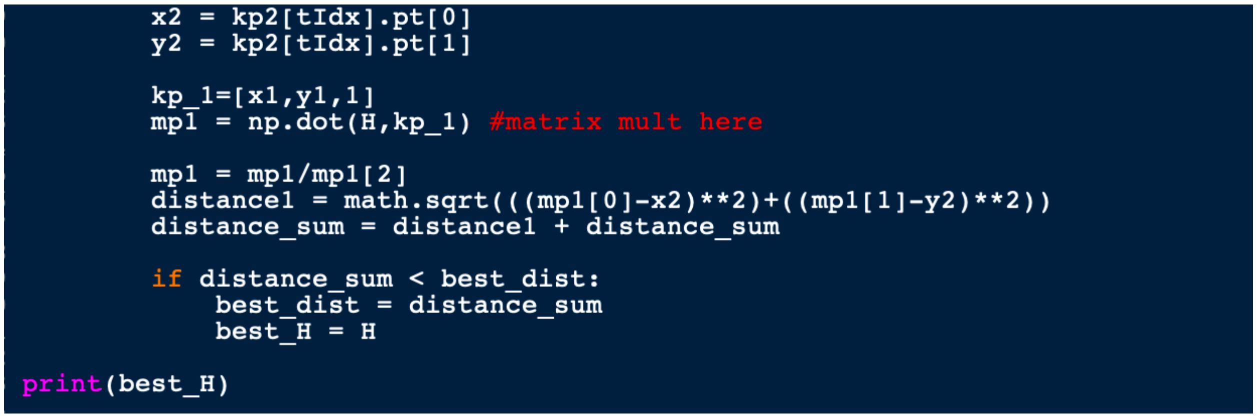
```python
import numpy as np
import cv2
from matplotlib import pyplot as plt
import random
import math

img1 = cv2.imread('mountainLeft.jpg') # queryImage
img2 = cv2.imread('mountainRight.jpg') # trainImage

im2_Rows = img2.shape[0]
im2_cols = img2.shape[1]

img1_Rows = img1.shape[0]
img1_Cols = img1.shape[1]

# Initiate SIFT detector
sift = cv2.SIFT_create()
##sift = cv2.xfeatures2d.SIFT_create()
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

# BFMatcher with default params
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

# Apply ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

# cv2.drawMatchesKnn expects list of lists as matches.
img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2) #NOTE: 'None'

plt.imshow(img3),plt.show()
```

Next, we have to create and run our own RANSAC algorithm which in theory is to pick the best Homography matrix that aligns the majority of the SIFT matches between the two images. The first step in the RANSAC creation would be to initialize variables to keep track of the best distance as well as set the number of iterations so that your Homography can have the lowest error. In each iteration we are picking four random matches/points from our “good matches” list that was created previously. We then create our A matrix. Then we run our SVD provided code as well as reformat the output of the SVD code so that the output is in a 3X3 Homography matrix instead of the original 8X9. All of this is shown in the picture below:



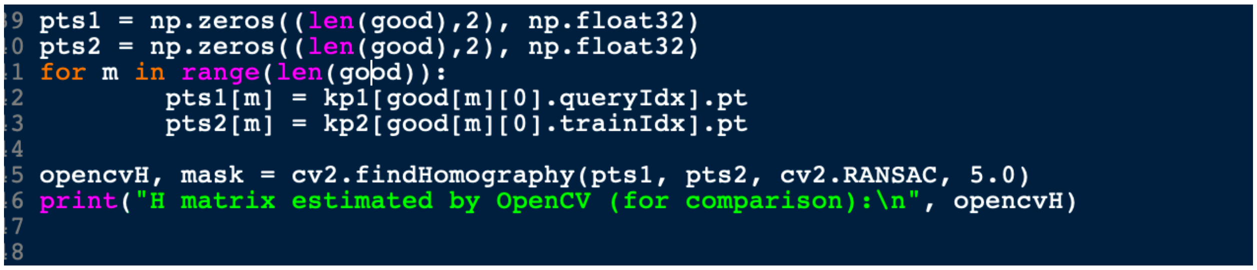
```
A = []
best_dist = 300000
for a in range(0,20000):
    for r in range(4):
        i = random.randint(0,len(good)-1)

        qIdx = good[i][0].queryIdx
        tIdx = good[i][0].trainIdx
        x1 = kp1[qIdx].pt[0]
        y1 = kp1[qIdx].pt[1]
        x2 = kp2[tIdx].pt[0]
        y2 = kp2[tIdx].pt[1]

        A.append([0,0,0,-x2,-y2,-1,(y1*x2),(y1*y2),-y1])
        A.append([x2,y2,1,0,0,0,(-x1*x2),(-x1*y2),-x1])


    U, s, V = np.linalg.svd(A, full_matrices=True)
    hCol = np.zeros((9,1), np.float64)
    hCol = V[8,:]

##print("MyMatrix",hCol/hCol[8])
    H = [[hCol[0],hCol[1],hCol[2]],[hCol[3],hCol[4],hCol[5]],[hCol[6],hCol[7
    distance_sum = 0
    for i in range(len(good)):

        qIdx = good[i][0].queryIdx
        tIdx = good[i][0].trainIdx
        x1 = kp1[qIdx].pt[0]
        y1 = kp1[qIdx].pt[1]
        x2 = kp2[tIdx].pt[0]
```

We are then going to look at every match in “good” and get the distance between our pt1 and pt2. Then after multiplying my H matrix by the pt1 in the “good’ matches we then divide through by the homogeneous coordinate as well as comparing the distance sum to the best distance to see if the distance sum is less than the best distance and if that is true then the new best distance is the current distance sum. We also then have to make our “best H” our new “H”. Then we would print out our best H.



```
        x2 = kp2[tIdx].pt[0]
        y2 = kp2[tIdx].pt[1]

        kp_1=[x1,y1,1]
        mp1 = np.dot(H,kp_1) #matrix mult here

        mp1 = mp1/mp1[2]
        distance1 = math.sqrt(((mp1[0]-x2)**2)+((mp1[1]-y2)**2))
        distance_sum = distance1 + distance_sum

        if distance_sum < best_dist:
            best_dist = distance_sum
            best_H = H

print(best_H)
```

We then are comparing our Homography Matrix that was created through our created RANSAC implementation to the created Homography Matrix that was created using OpenCV's "findHomography" function to compare how well our matches actually were.

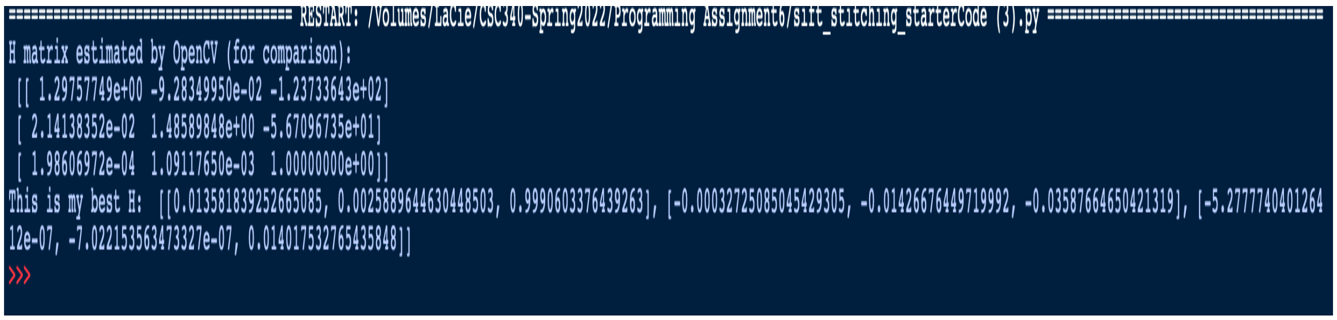


```python
pts1 = np.zeros((len(good),2), np.float32)
pts2 = np.zeros((len(good),2), np.float32)
for m in range(len(good)):
        pts1[m] = kp1[good[m][0].queryIdx].pt
        pts2[m] = kp2[good[m][0].trainIdx].pt

opencvH, mask = cv2.findHomography(pts1, pts2, cv2.RANSAC, 5.0)
print("H matrix estimated by OpenCV (for comparison):\n", opencvH)


```

Now with all of that being said here are the comparison between my RANSAC implementation and OpenCV's implementation showing the Homography Matrix that was computed as the best.
Disclaimer!: My matches will not be as close or as accurate as they could've been due to a difficulty I was having with the elongated runtime that my code was having which was over 5+ hours.

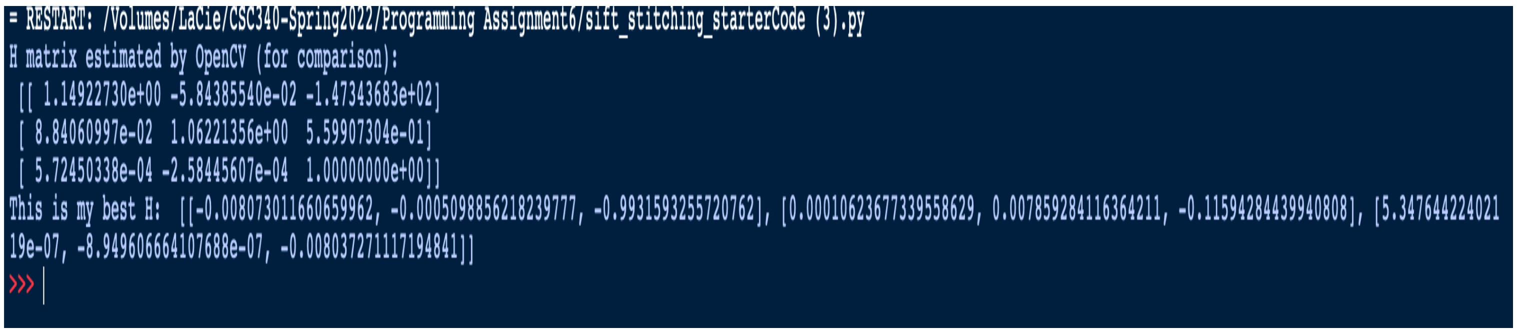
Now, with that being said here are all of the Homography Matrix comparisons by picture:

Florence:



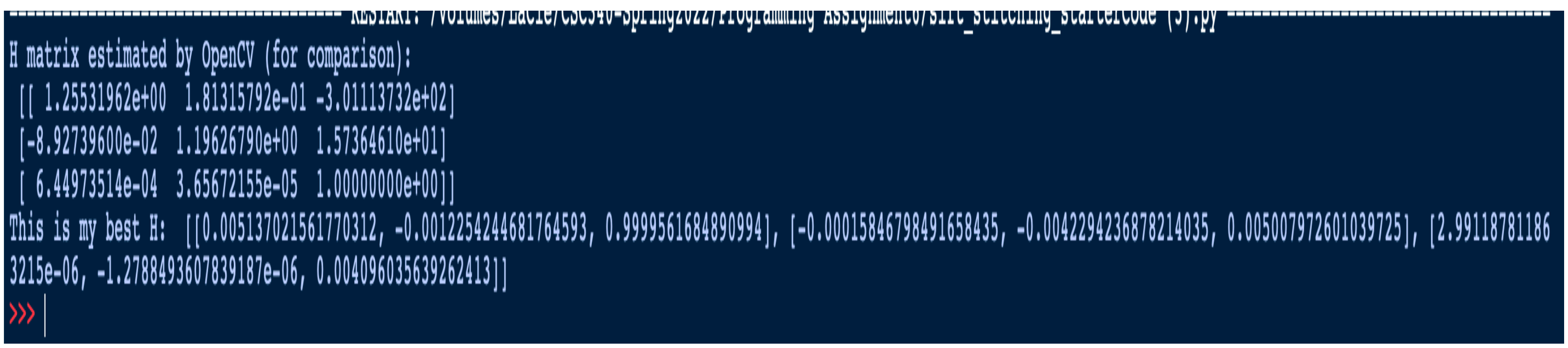
```
==================================== RESTART: /Volumes/LaCie/CSC340-Spring2022/Programming Assignment6/sift_stitching_starterCode (3).py ====================================
H matrix estimated by OpenCV (for comparison):
 [[ 1.29757749e+00 -9.28349950e-02 -1.23733643e+02]
 [ 2.14138352e-02  1.48589848e+00 -5.67096735e+01]
 [ 1.98606972e-04  1.09117650e-03  1.00000000e+00]]
This is my best H:  [[0.013581839252665085, 0.0025889644630448503, 0.9990603376439263], [-0.00032725085045429305, -0.01426676449719992, -0.03587664650421319], [-5.277774040126412e-07, -7.022153563473327e-07, 0.014017532765435848]]
>>>
```

Mountains:



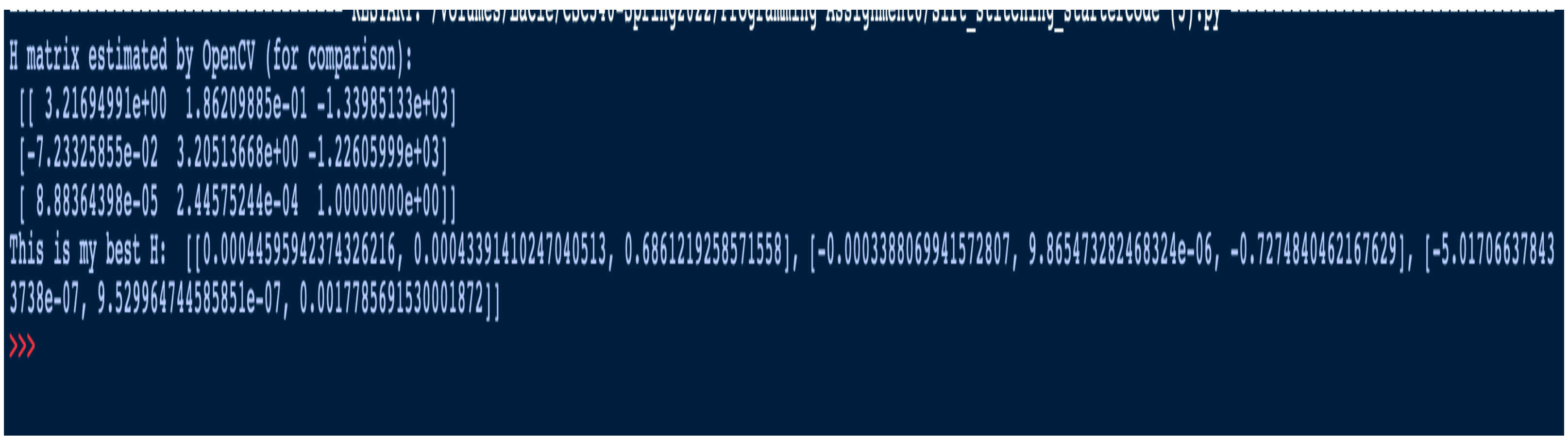
```
= RESTART: /Volumes/LaCie/CSC340-Spring2022/Programming Assignment6/sift_stitching_starterCode (3).py
H matrix estimated by OpenCV (for comparison):
 [[ 1.14922730e+00 -5.84385540e-02 -1.47343683e+02]
 [ 8.84060997e-02  1.06221356e+00  5.59907304e-01]
 [ 5.72450338e-04 -2.58445607e-04  1.00000000e+00]]
This is my best H:  [[-0.008073011660659962, -0.0005098856218239777, -0.9931593255720762], [0.00010623677339558629, 0.007859284116364211, -0.11594284439940808], [5.34764422402119e-07, -8.949606664107688e-07, -0.008037271117194841]]
>>> 
```

Queenstown:



```
======================================= RESTART: /Volumes/LaCie/CSC340-Spring2022/Programming Assignments/sift_stitching_startercode (5).py =======================================
H matrix estimated by OpenCV (for comparison):
 [[ 1.25531962e+00  1.81315792e-01 -3.01113732e+02]
 [-8.92739600e-02  1.19626790e+00  1.57364610e+01]
 [ 6.44973514e-04  3.65672155e-05  1.00000000e+00]]
This is my best H:  [[0.005137021561770312, -0.0012254244681764593, 0.9999561684890994], [-0.00015846798491658435, -0.0042294236878214035, 0.005007972601039725], [2.99118781186
3215e-06, -1.2788493607839187e-06, 0.004096035639262413]]
>>> 
```

Waterfall:



```
======================================= RESTART: /Volumes/LaCie/CSC340-Spring2022/Programming Assignments/sift_stitching_startercode (5).py =======================================
H matrix estimated by OpenCV (for comparison):
 [[ 3.21694991e+00  1.86209885e-01 -1.33985133e+03]
 [-7.23325855e-02  3.20513668e+00 -1.22605999e+03]
 [ 8.88364398e-05  2.44575244e-04  1.00000000e+00]]
This is my best H:  [[0.00044595942374326216, 0.00043391410247040513, 0.6861219258571558], [-0.0003388069941572807, 9.865473282468324e-06, -0.7274840462167629], [-5.01706637843
3738e-07, 9.529964744585851e-07, 0.0017785691530001872]]
>>> 
```