Mohanty_R_HW1_Prob2_1

February 20, 2020

1 2. Computing

1. Write a code to generate 1000 random numbers contained on the unit circle. Apply several random matrices to this data and describe your results in the terminology of bases and change of bases. How do your results differ if the multiplying matrix is constrained to be orthogonal?

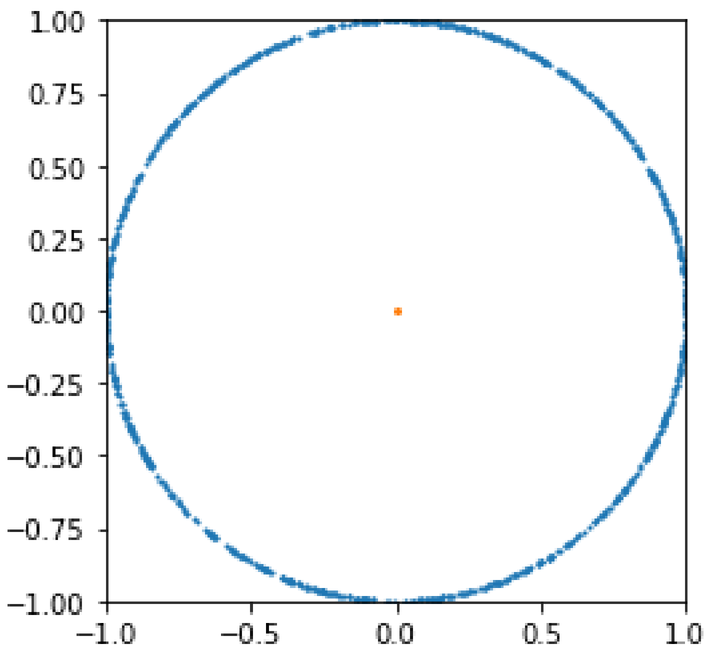
Importing required python packages

[15]:
```
import random
import math
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Genrating random vectors on a unit circle

[13]:
```
unit_circle_points=[]
for i in range(0,1000):
    theta=random.uniform(0,2*math.pi)
    tup=(math.cos(theta),math.sin(theta))
    unit_circle_points.append(tup)
```

[34]:
```
plt.scatter(*zip(*unit_circle_points),1)
plt.scatter(0,0,4)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



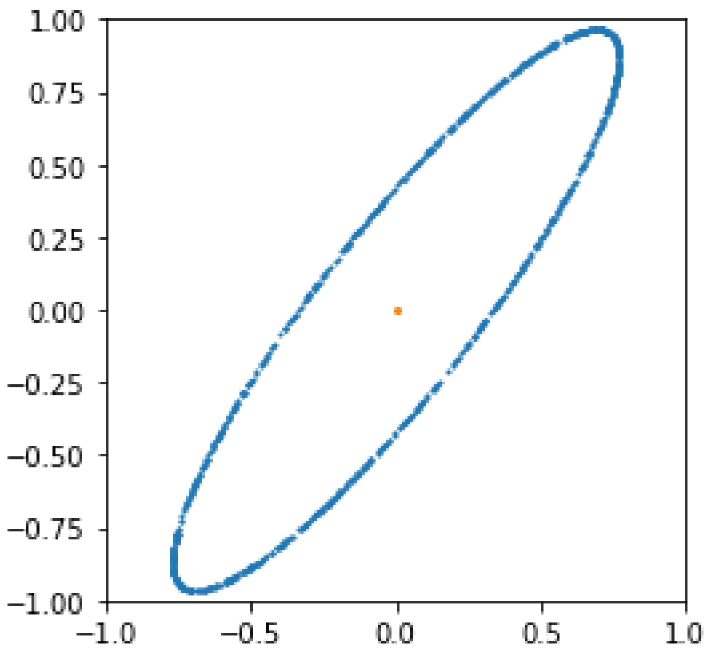
Generating 10 random transformation matrices

[42]:
```
transf_random_matrix={}
random_matrix_list=[]
for i in range(0,10):
    random_matrix=np.random.rand(2,2)
    random_matrix_list.append(random_matrix)
    temp_list=[]
    for j in range(0,1000):
        vec=np.array(unit_circle_points[j])
        vec_newb=tuple(np.matmul(random_matrix,vec))
        temp_list.append(vec_newb)
    transf_random_matrix.update({i:temp_list})
```

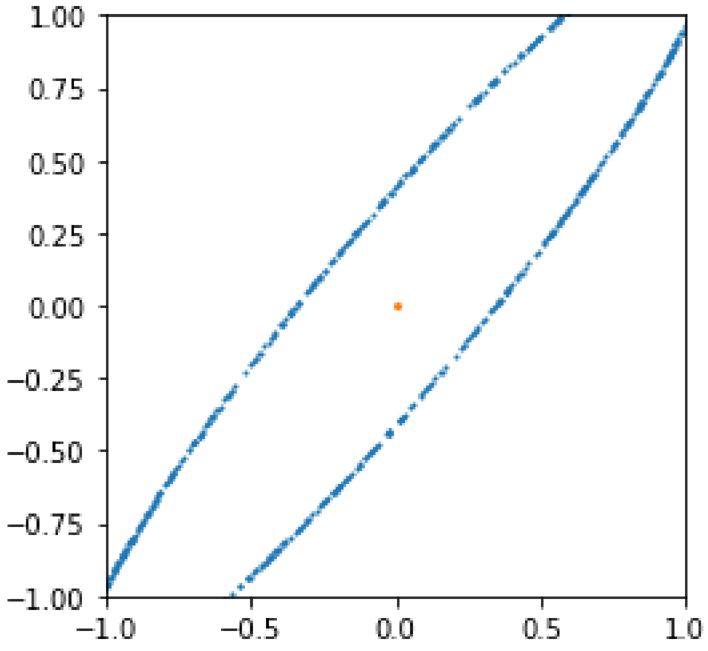
Applying the random matrices to the points on unit circle generated previously

[46]:
```
for i in range(0,10):
    print("Random transformation matrix: ")
    print(random_matrix_list[i])
    print("Resulting vectors: ")
    plt.scatter(*zip(*transf_random_matrix[i]),1)
    plt.scatter(0,0,4)
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()
```

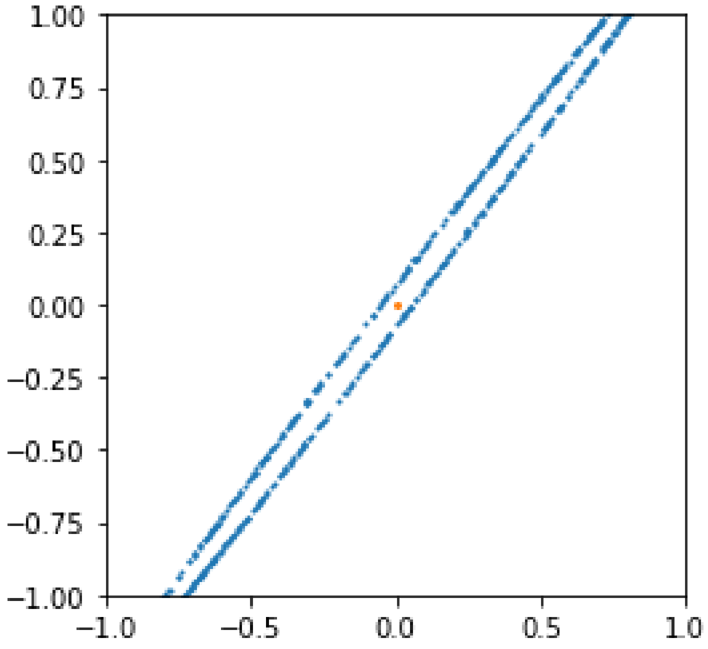
```
Random transformation matrix:
[[0.22575337 0.73518422]
 [0.65822262 0.70565941]]
Resulting vectors:
```



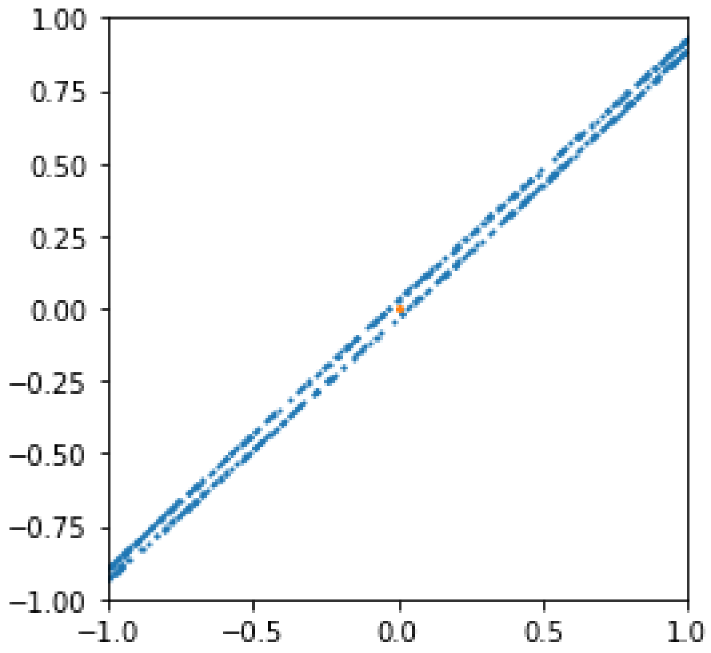
```
Random transformation matrix:
[[0.93454867 0.56459857]
 [0.84607731 0.98905391]]
Resulting vectors:
```



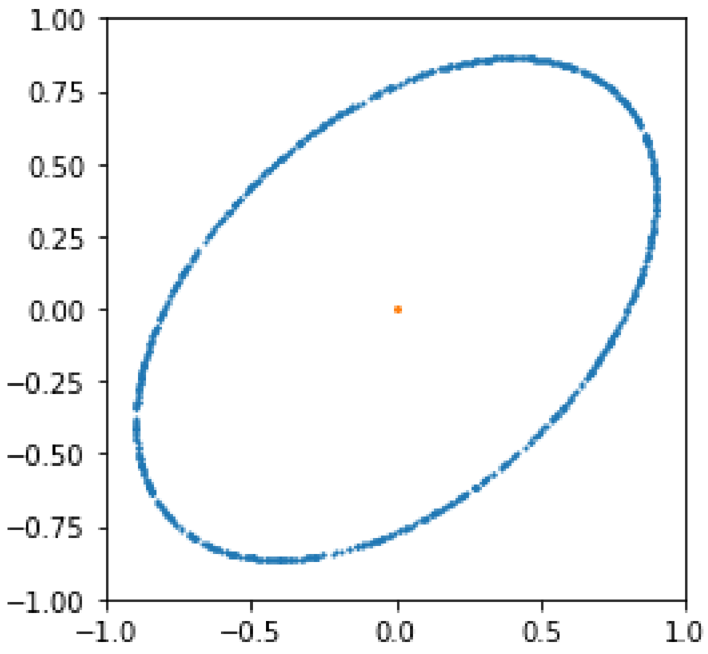
```
Random transformation matrix:
[[0.66249002 0.7337029 ]
 [0.92030161 0.91231951]]
Resulting vectors:
```



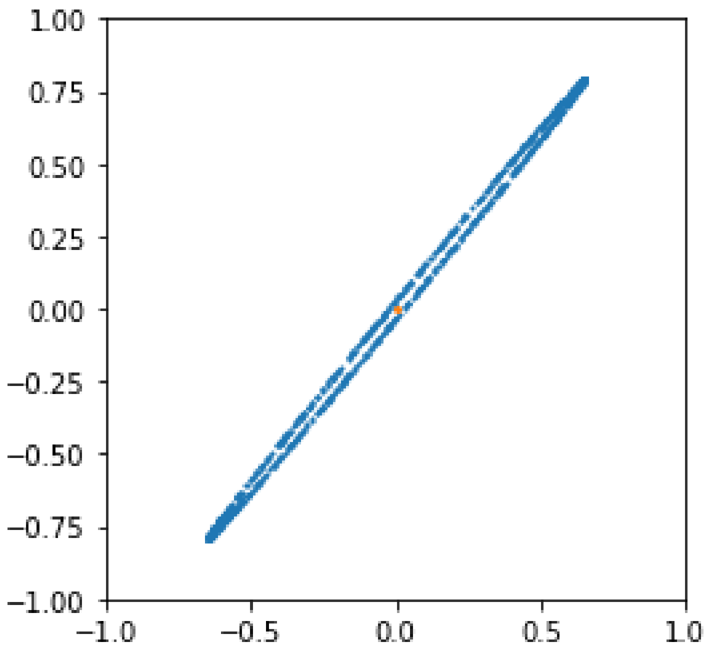
```
Random transformation matrix:
[[0.93212042 0.85975177]
 [0.83033423 0.80572047]]
Resulting vectors:
```



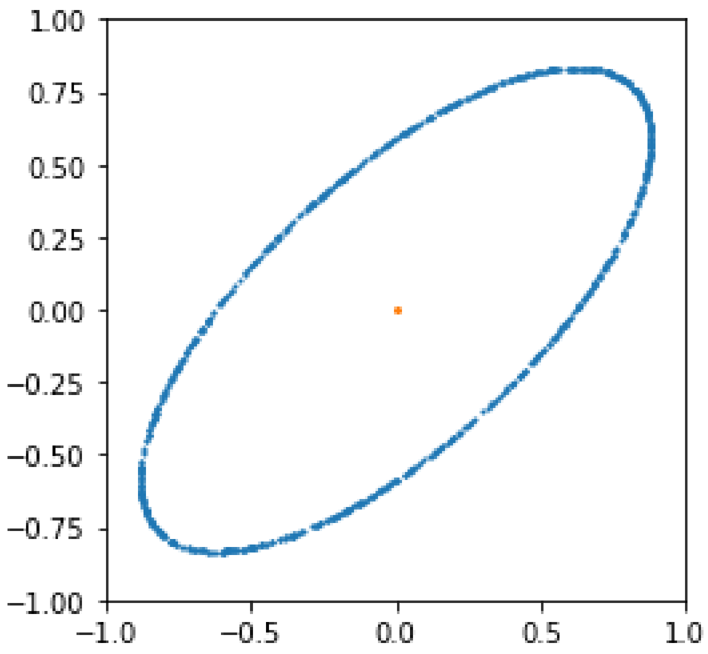
```
Random transformation matrix:
[[0.87643628 0.18585203]
 [0.22244176 0.83504171]]
Resulting vectors:
```



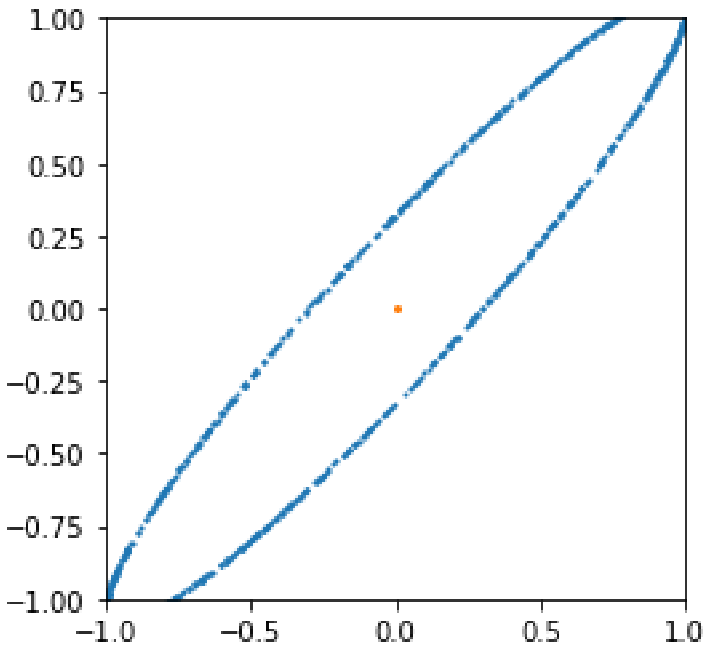
```
Random transformation matrix:
[[0.06995748 0.64614932]
 [0.05350939 0.79047615]]
Resulting vectors:
```



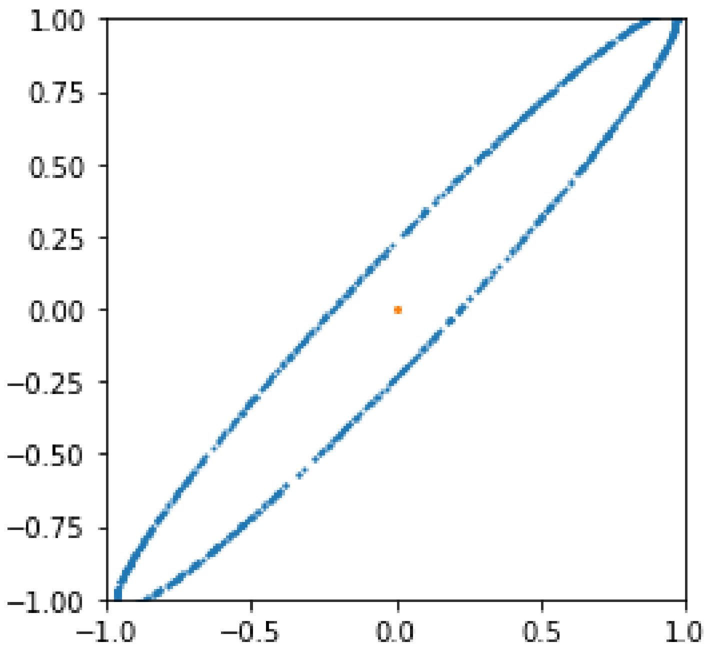
```
Random transformation matrix:
[[0.15978196 0.86461301]
 [0.6857766  0.47246319]]
Resulting vectors:
```



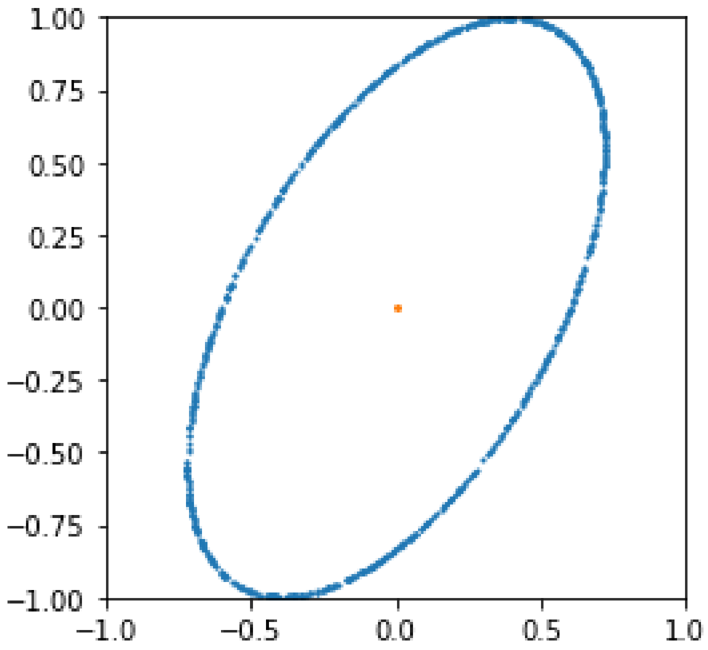
```
Random transformation matrix:
[[0.47954722 0.87699376]
 [0.78050054 0.75092323]]
Resulting vectors:
```



```
Random transformation matrix:
[[0.91407212 0.31133326]
 [0.87044455 0.54420342]]
Resulting vectors:
```



```
Random transformation matrix:
[[0.06878169 0.71579913]
 [0.88223286 0.46569709]]
Resulting vectors:
```



1.1 We notice that these random matrices which most likely are not orthonormal bases change the angle (between vectors) and length of vectors. Next we will see how orthonormal bases preserve angle and length

This function generates random Orthnormal matrices

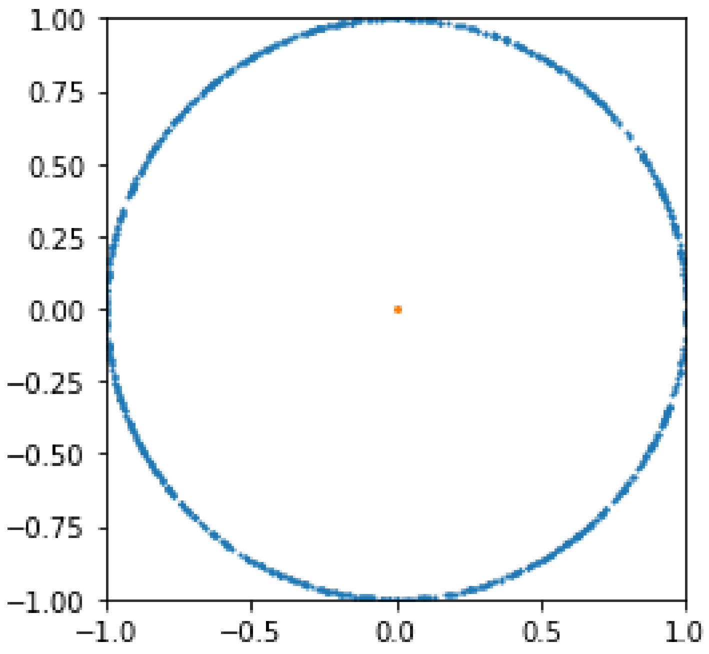
[48]:
```python
def rvs(dim=3):
    random_state = np.random
    H = np.eye(dim)
    D = np.ones((dim,))
    for n in range(1, dim):
        x = random_state.normal(size=(dim-n+1,))
        D[n-1] = np.sign(x[0])
        x[0] -= D[n-1]*np.sqrt((x*x).sum())
        # Householder transformation
        Hx = (np.eye(dim-n+1) - 2.*np.outer(x, x)/(x*x).sum())
        mat = np.eye(dim)
        mat[n-1:, n-1:] = Hx
        H = np.dot(H, mat)
        # Fix the last sign such that the determinant is 1
    D[-1] = (-1)**(1-(dim % 2))*D.prod()
    # Equivalent to np.dot(np.diag(D), H) but faster, apparently
    H = (D*H.T).T
    return H
```

generate 10 random orthnormal matrices and apply it to the random points on unit circle

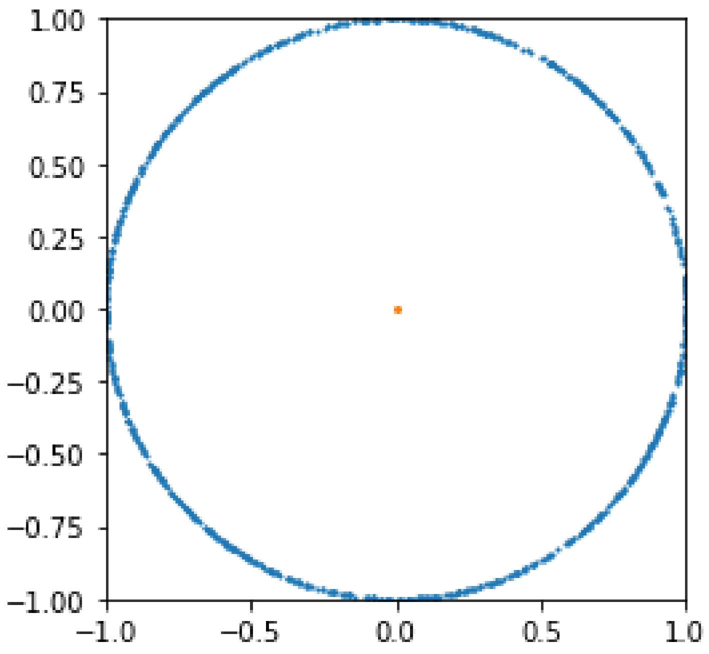
[53]:
```
transf_random_orthonormal_matrices={}
random_orthn_matrix_list=[]
for i in range(0,10):
    random_ortho_matrix=rvs(dim=2)
    random_orthn_matrix_list.append(random_ortho_matrix)
    temp_list=[]
    for j in range(0,1000):
        vec=np.array(unit_circle_points[j])
        vec_newb=tuple(np.matmul(random_ortho_matrix,vec))
        temp_list.append(vec_newb)
    transf_random_orthonormal_matrices.update({i:temp_list})
```

[54]:
```
for i in range(0,10):
    print("Random orthonormal transformation matrix: ")
    print(random_orthn_matrix_list[i])
    print("Resulting vectors: ")
    plt.scatter(*zip(*transf_random_orthonormal_matrices[i]),1)
    plt.scatter(0,0,4)
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()
```

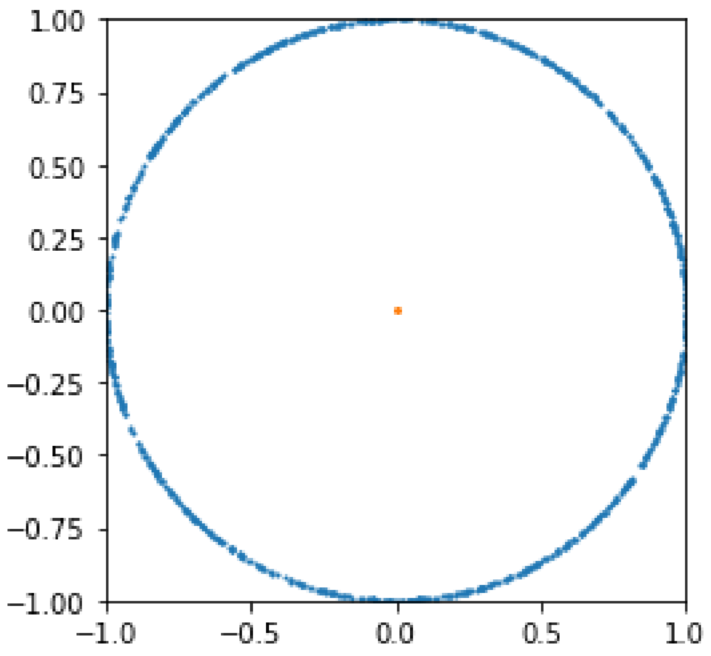
```
Random orthonormal transformation matrix:
[[-0.61508484 -0.78846093]
 [ 0.78846093 -0.61508484]]
Resulting vectors:
```



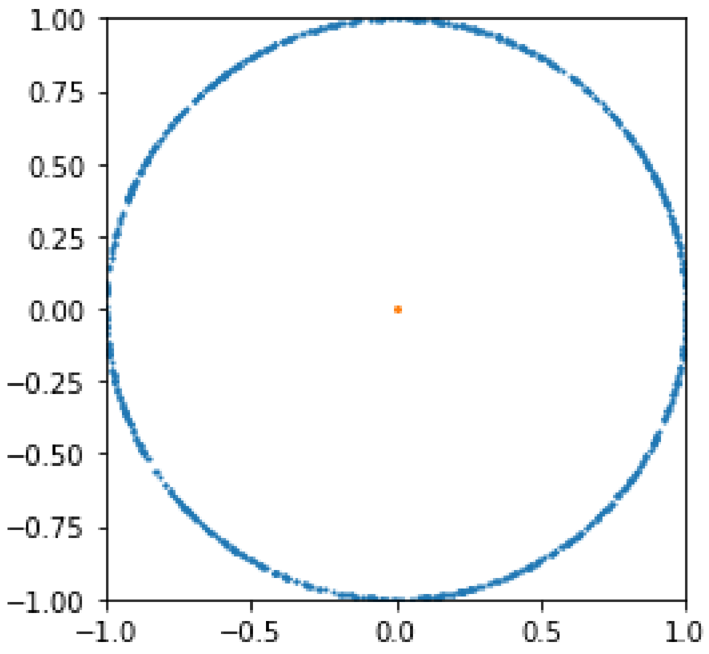
```
Random orthonormal transformation matrix:
[[-0.89972852 -0.43644999]
 [ 0.43644999 -0.89972852]]
Resulting vectors:
```



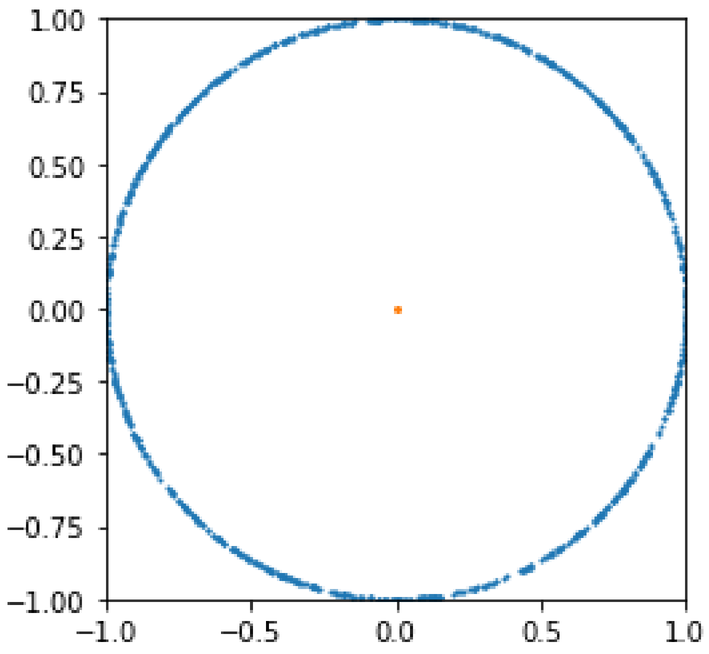
```
Random orthonormal transformation matrix:
[[ 0.43115435  0.90227819]
 [-0.90227819  0.43115435]]
Resulting vectors:
```



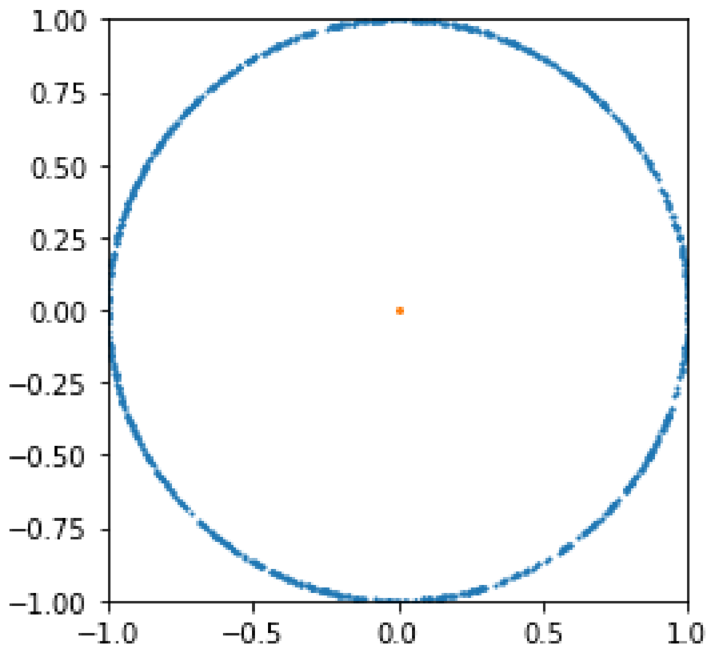
```
Random orthonormal transformation matrix:
[[ 0.58213571  0.81309164]
 [-0.81309164  0.58213571]]
Resulting vectors:
```



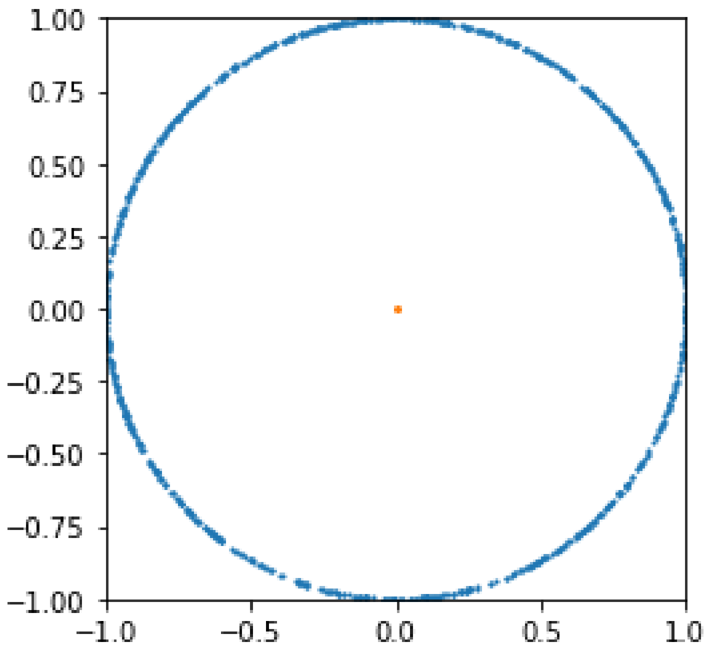
```
Random orthonormal transformation matrix:
[[ 0.40250507 -0.91541776]
 [ 0.91541776  0.40250507]]
Resulting vectors:
```



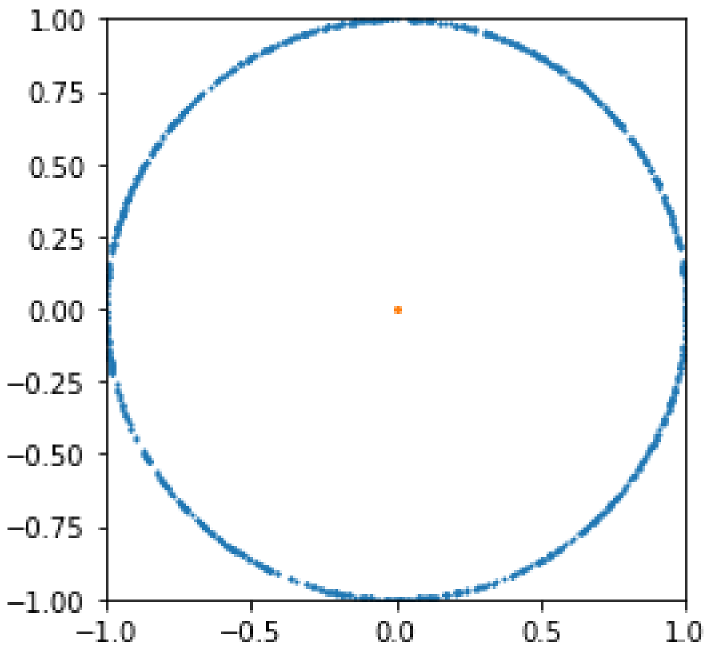
```
Random orthonormal transformation matrix:
[[ 0.26387908 -0.96455577]
 [ 0.96455577  0.26387908]]
Resulting vectors:
```



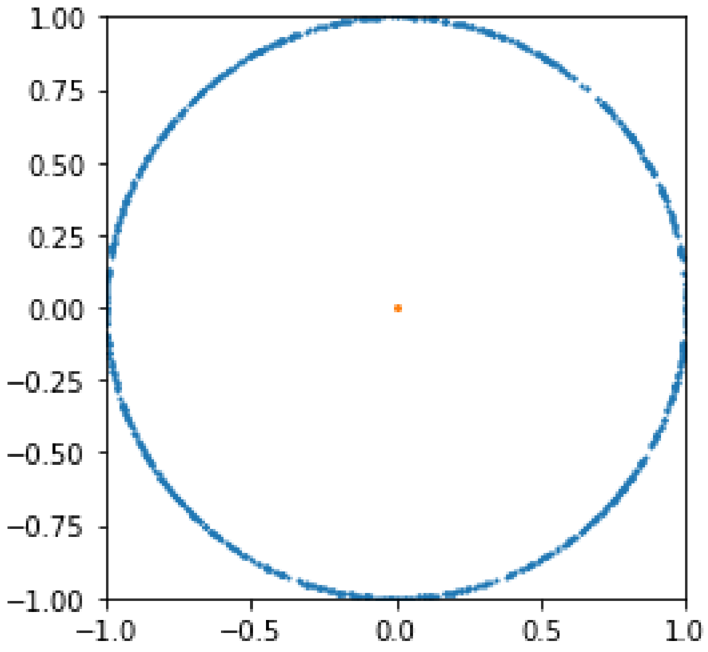
```
Random orthonormal transformation matrix:
[[ 0.9356732  -0.35286778]
 [ 0.35286778  0.9356732 ]]
Resulting vectors:
```



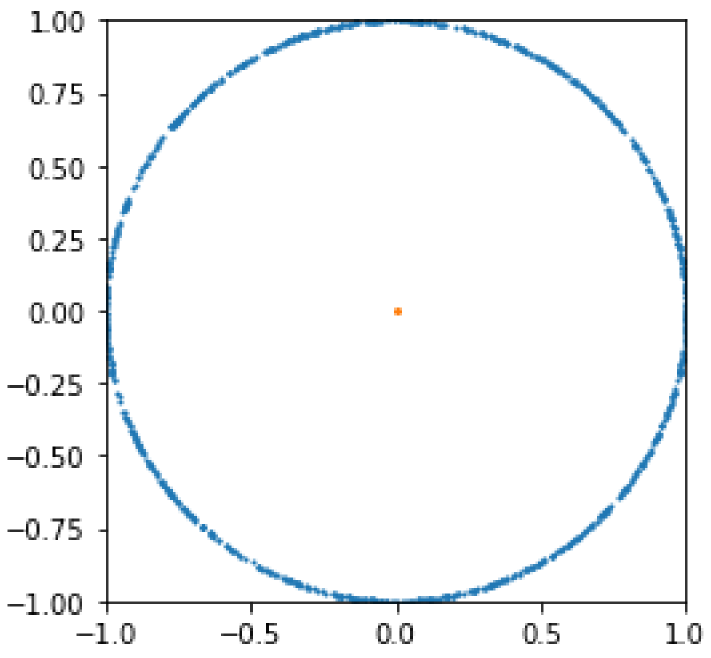
```
Random orthonormal transformation matrix:
[[ 0.94061455  0.33947646]
 [-0.33947646  0.94061455]]
Resulting vectors:
```



```
Random orthonormal transformation matrix:
[[-0.77161528 -0.63608951]
 [ 0.63608951 -0.77161528]]
Resulting vectors:
```



```
Random orthonormal transformation matrix:
[[ 0.31471023  0.9491878 ]
 [-0.9491878   0.31471023]]
Resulting vectors:
```



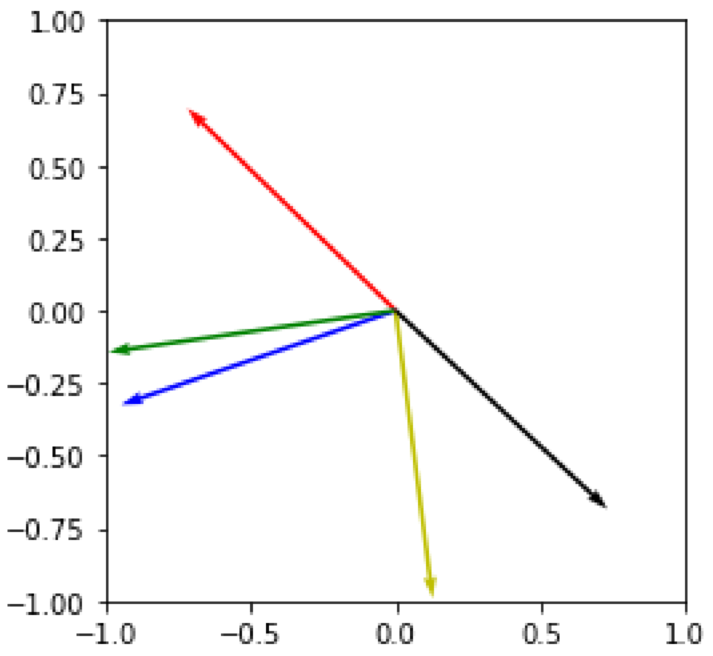
As expected the vector are still alligned on the unit circle prooving that the orthonormal change of basis preserves angle and length

The below code and vizualization show how orthnormal change of basis preserves angle and length where as random matrices change angle and lengths. He 5 random vectors on the unit circle were chosen.

[83]:
```
random.shuffle(unit_circle_points)
```

[84]:
```
random_5_vectors=unit_circle_points[0:5]
```

[91]:
```
origin = [0], [0] # origin point

plt.quiver(*origin, *zip(*random_5_vectors), color=['r','b','g','y','k'],␣
 ↪scale=2)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



[93]:
```
random_orthn_matrix_list[0]
```

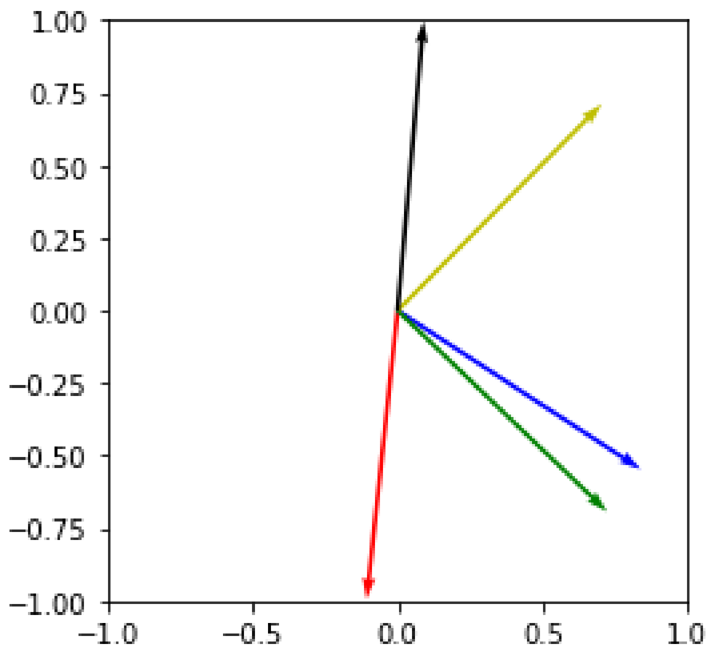
[93]:
```
array([[-0.61508484, -0.78846093],
       [ 0.78846093, -0.61508484]])
```

[94]:
```
random_5_vectors
```

[94]:
```
[(-0.718264331677728, 0.6957703283696041),
 (-0.9459548320233664, -0.32429840544110705),
 (-0.989722323441586, -0.14300252613638925),
 (0.1277629946573041, -0.9918047273512048),
 (0.7300997226815352, -0.6833406141452045)]
```

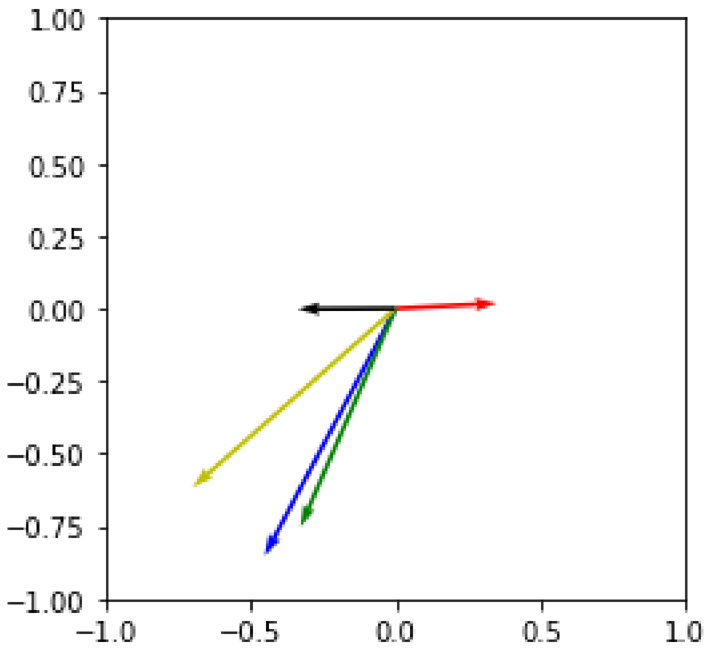
[95]:
```
orthonormal_trans_5vectors=[]
for j in range(0,5):
    vec=np.array(random_5_vectors[j])
    vec_newb=tuple(np.matmul(random_orthn_matrix_list[0],vec))
    orthonormal_trans_5vectors.append(vec_newb)
```

[96]:
```
origin = [0], [0] # origin point

plt.quiver(*origin, *zip(*orthonormal_trans_5vectors),␣
 ↪color=['r','b','g','y','k'], scale=2)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



[97]:
```
random_trans_5vectors=[]
for j in range(0,5):
    vec=np.array(random_5_vectors[j])
    vec_newb=tuple(np.matmul(random_matrix_list[0],vec))
    random_trans_5vectors.append(vec_newb)
```

[98]:
```
origin = [0], [0] # origin point

plt.quiver(*origin, *zip(*random_trans_5vectors), color=['r','b','g','y','k'],␣
 ↪scale=2)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0

[ ]:

```

```