Begin by installing required packages

In [16]:
```
#install MPHvect from the repository

!pip install git+https://github.com/ZachTRoss/MPHvect.git

#install other tools used for this example file
!pip install chromatic_tda
!pip install ripser

!pip install scikit-learn

#MPHvect works faster with numba installed
!pip install numba
```

```
Collecting git+https://github.com/ZachTRoss/MPHvect.git
  Cloning https://github.com/ZachTRoss/MPHvect.git to /tmp/pip-req-build-nhgot8we
  Running command git clone --filter=blob:none --quiet https://github.com/ZachTRoss/MPHvect.git
/tmp/pip-req-build-nhgot8we
  Resolved https://github.com/ZachTRoss/MPHvect.git to commit 8ebd95d34efd44e4890ff01babe813626a2c89b7
  Preparing metadata (setup.py) ... ?25l?2518[?25hdone
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from MPHvect==0.1.0)
(2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from
MPHvect==0.1.0) (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (from
MPHvect==0.1.0) (1.6.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages (from MPHvect==0.1.0)
(5.24.1)
Requirement already satisfied: gudhi in /usr/local/lib/python3.12/dist-packages (from MPHvect==0.1.0)
(3.11.0)
Requirement already satisfied: ripser in /usr/local/lib/python3.12/dist-packages (from MPHvect==0.1.0)
(0.6.14)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->MPHvect==0.1.0) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->MPHvect==0.1.0) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->MPHvect==0.1.0) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->MPHvect==0.1.0) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->MPHvect==0.1.0) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib-
>MPHvect==0.1.0) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->MPHvect==0.1.0) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->MPHvect==0.1.0) (2.9.0.post0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from
plotly->MPHvect==0.1.0) (9.1.2)
Requirement already satisfied: Cython in /usr/local/lib/python3.12/dist-packages (from ripser-
>MPHvect==0.1.0) (3.0.12)
Requirement already satisfied: persim in /usr/local/lib/python3.12/dist-packages (from ripser-
>MPHvect==0.1.0) (0.3.8)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from ripser-
>MPHvect==0.1.0) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-
learn->MPHvect==0.1.0) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from
scikit-learn->MPHvect==0.1.0) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-
dateutil>=2.7->matplotlib->MPHvect==0.1.0) (1.17.0)
Requirement already satisfied: deprecated in /usr/local/lib/python3.12/dist-packages (from persim-
>ripser->MPHvect==0.1.0) (1.3.1)
Requirement already satisfied: hopcroftkarp in /usr/local/lib/python3.12/dist-packages (from persim-
>ripser->MPHvect==0.1.0) (1.2.5)
Requirement already satisfied: wrapt<3,>=1.10 in /usr/local/lib/python3.12/dist-packages (from
deprecated->persim->ripser->MPHvect==0.1.0) (2.0.1)
Requirement already satisfied: chromatic_tda in /usr/local/lib/python3.12/dist-packages (1.1.8)
Requirement already satisfied: matplotlib>=3.8.0 in /usr/local/lib/python3.12/dist-packages (from
chromatic_tda) (3.10.0)
Requirement already satisfied: mypy<2.0.0,>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from
chromatic_tda) (1.19.1)
Collecting numpy<3.0,>=2.1 (from chromatic_tda)
  Using cached numpy-2.4.1-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (6.6
kB)
Requirement already satisfied: pytest<9.0.0,>=8.1.1 in /usr/local/lib/python3.12/dist-packages (from
chromatic_tda) (8.4.2)
Requirement already satisfied: scipy<2.0,>=1.14 in /usr/local/lib/python3.12/dist-packages (from
chromatic_tda) (1.16.3)
Requirement already satisfied: typing_extensions<5.0.0,>=4.11.0 in /usr/local/lib/python3.12/dist-
packages (from chromatic_tda) (4.15.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from
matplotlib>=3.8.0->chromatic_tda) (2.9.0.post0)
Requirement already satisfied: mypy_extensions>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from
mypy<2.0.0,>=1.3.0->chromatic_tda) (1.1.0)
Requirement already satisfied: pathspec>=0.9.0 in /usr/local/lib/python3.12/dist-packages (from
mypy<2.0.0,>=1.3.0->chromatic_tda) (1.0.3)
Requirement already satisfied: librt>=0.6.2 in /usr/local/lib/python3.12/dist-packages (from
mypy<2.0.0,>=1.3.0->chromatic_tda) (0.7.8)
Requirement already satisfied: iniconfig>=1 in /usr/local/lib/python3.12/dist-packages (from
pytest<9.0.0,>=8.1.1->chromatic_tda) (2.3.0)
Requirement already satisfied: pluggy<2,>=1.5 in /usr/local/lib/python3.12/dist-packages (from
pytest<9.0.0,>=8.1.1->chromatic_tda) (1.6.0)
Requirement already satisfied: pygments>=2.7.2 in /usr/local/lib/python3.12/dist-packages (from
pytest<9.0.0,>=8.1.1->chromatic_tda) (2.19.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-
dateutil>=2.7->matplotlib>=3.8.0->chromatic_tda) (1.17.0)
Using cached numpy-2.4.1-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.4 MB)
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
?[31mERROR: pip's dependency resolver does not currently take into account all the packages that are
installed. This behaviour is the source of the following dependency conflicts.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 2.4.1
which is incompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.4.1 which is incompatible.
tensorflow 2.19.0 requires numpy<2.2.0,>=1.26.0, but you have numpy 2.4.1 which is incompatible.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy
2.4.1 which is incompatible.
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy
2.4.1 which is incompatible.?[0m?[31m
?[0mSuccessfully installed numpy-2.4.1
```

```
Requirement already satisfied: ripser in /usr/local/lib/python3.12/dist-packages (0.6.14)
Requirement already satisfied: Cython in /usr/local/lib/python3.12/dist-packages (from ripser)
(3.0.12)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from ripser) (2.4.1)
Requirement already satisfied: persim in /usr/local/lib/python3.12/dist-packages (from ripser) (0.3.8)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from ripser) (1.16.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (from ripser)
(1.6.1)
Requirement already satisfied: deprecated in /usr/local/lib/python3.12/dist-packages (from persim-
>ripser) (1.3.1)
Requirement already satisfied: hopcroftkarp in /usr/local/lib/python3.12/dist-packages (from persim-
>ripser) (1.2.5)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from persim->ripser)
(1.5.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from persim-
>ripser) (3.10.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from
scikit-learn->ripser) (3.6.0)
Requirement already satisfied: wrapt<3,>=1.10 in /usr/local/lib/python3.12/dist-packages (from
deprecated->persim->ripser) (2.0.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->persim->ripser) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->persim->ripser) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->persim->ripser) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->persim->ripser) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->persim->ripser) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib-
>persim->ripser) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->persim->ripser) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from
matplotlib->persim->ripser) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-
dateutil>=2.7->matplotlib->persim->ripser) (1.17.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-
learn) (2.4.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-
learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-
learn) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from
scikit-learn) (3.6.0)
Requirement already satisfied: numba in /usr/local/lib/python3.12/dist-packages (0.60.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages
(from numba) (0.43.0)
```

```
Collecting numpy<2.1,>=1.22 (from numba)
  Using cached numpy-2.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
Using cached numpy-2.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (19.2 MB)
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.4.1
    Uninstalling numpy-2.4.1:
      Successfully uninstalled numpy-2.4.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are
installed. This behaviour is the source of the following dependency conflicts.
chromatic-tda 1.1.0 requires numpy<3.0,>=2.1, but you have numpy 2.0.2 which is incompatible.
Successfully installed numpy-2.0.2
```

In [1]:
```
import numba
import numpy as np
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors



from ripser import ripser
from persim import plot_diagrams
import itertools
from chromatic_tda import ChromaticAlphaComplex
import random
import chromatic_tda as chro
from mpl_toolkits.mplot3d import Axes3D

import numpy as np
import sys, math




from MPHvect.method import*


import glob
import gc
import os
import IPython
import subprocess

import gudhi as gd
from gudhi.representations import Landscape
```

First we define functions. I put the function for calculating mixup barcodes into a single function. The rest of these functions are for sampling methods.

In [2]:
```
# @title




def Calculate_Mixup(points1, points2):
  points=np.vstack((points1, points2))

  labels1 = [0 for _ in range(len(points1))]
  labels2=  [1 for _ in range(len(points2))]
  labels=labels1+labels2

  alpha = ChromaticAlphaComplex(points=points, labels=labels)
  del points, labels

  simplicial_complex = alpha.get_simplicial_complex(sub_complex='0')
  del alpha

  simplicial_complex.compute_persistence()
  feature_extractor = chro.experimental.feature_extraction.FeatureExtractor(simplicial_complex)

  sub_pairs = feature_extractor.persistence_pairs(group='sub_complex', dim=1, sorted_by='persistence')
  im_pairs = feature_extractor.persistence_pairs(group='image', dim=1, sorted_by='persistence')

      # --- Convert to dictionary ---
  im_dth = dict(im_pairs)
  del im_pairs

  w = simplicial_complex.weight_function

    # --- Allocate output ---
  n = len(sub_pairs)
  mixup_barcode = np.empty((n, 3), dtype=np.float32)

  for i, (b, d) in enumerate(sub_pairs):
      wb = w(b)
      wd = w(d)

      if b in im_dth:
          mixup_barcode[i, 0] = wb
          mixup_barcode[i, 1] = w(im_dth[b])
          mixup_barcode[i, 2] = wd
      else:
          mixup_barcode[i, 0] = wb
          mixup_barcode[i, 1] = wd
          mixup_barcode[i, 2] = wd

    # --- Explicit cleanup ---
  del sub_pairs, im_dth, feature_extractor, simplicial_complex
  gc.collect()  # optional but helpful in long loops

  return mixup_barcode


#Sampling Methods

def sample_annulus(
    n_points,
    center=(0.0, 0.0),
    r_inner=1.0,
    r_outer=2.0,
    random_state=None
):
    """
    Sample points uniformly from a 2D annulus.

    Parameters
    ----------
    n_points : int
        Number of points to sample.
    center : tuple of float
        (x, y) center of the annulus.
    r_inner : float
        Inner radius (>= 0).
    r_outer : float
        Outer radius (> r_inner).
    random_state : int or None
        Seed for reproducibility.

    Returns
    -------
    points : ndarray, shape (n_points, 2)
        Sampled points.
    """
    if r_inner < 0 or r_outer <= r_inner:
        raise ValueError("Require 0 <= r_inner < r_outer")

    rng = np.random.default_rng(random_state)

    # Sample angle uniformly
    theta = rng.uniform(0.0, 2.0 * np.pi, size=n_points)

    # Sample radius with area correction
    u = rng.uniform(0.0, 1.0, size=n_points)
    r = np.sqrt((r_outer**2 - r_inner**2) * u + r_inner**2)

    x = center[0] + r * np.cos(theta)
    y = center[1] + r * np.sin(theta)

    return np.column_stack((x, y))


def sample_disc(
        n_points,
        center=(0.0, 0.0),
        radius=0.5,
        random_state=None
    ):
        """
        Sample points uniformly from a 2D disc.

        Parameters
        ----------
        n_points : int
            Number of points to sample.
        center : tuple of float
            (x, y) center of the annulus.
        r_inner : float
            Inner radius (>= 0).
        r_outer : float
            Outer radius (> r_inner).
        random_state : int or None
            Seed for reproducibility.

        Returns
        -------
        points : ndarray, shape (n_points, 2)
            Sampled points.
        """
```

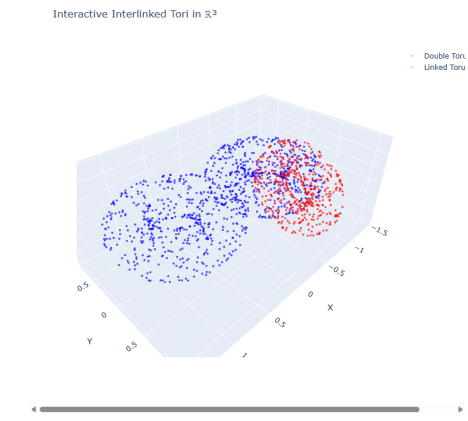
```
        if radius <0:
            raise ValueError("Require 0 <= radius")

        rng = np.random.default_rng(random_state)

        # Sample angle uniformly
        theta = rng.uniform(0.0, 2.0 * np.pi, size=n_points)

        # Sample radius with area correction
        u = rng.uniform(0.0, 1.0, size=n_points)
        r = radius * u

        x = center[0] + r * np.cos(theta)
        y = center[1] + r * np.sin(theta)

        return np.column_stack((x, y))



def sample_inside_annulus( disc_radius,annulus_inner_radius, annulus_outer_radius, n_points):

  n_annulus = math.ceil(n_points*((annulus_outer_radius)**2-
(annulus_inner_radius)**2)/((annulus_outer_radius)**2-(annulus_inner_radius)**2 + disc_radius**2))
  n_disc = n_points - n_annulus

  rng = np.random.default_rng()

  annulus_center_x = rng.uniform(-1.2,1.2)
  annulus_center_y = rng.uniform(-1.2,1.2)

  annulus_points=sample_annulus(n_annulus, center=(annulus_center_x,annulus_center_y),
r_inner=annulus_inner_radius, r_outer=annulus_outer_radius)

  disc_r_shift=rng.uniform(0,annulus_inner_radius-disc_radius)
  disc_theta_shift=rng.uniform(0,2*np.pi)

  disc_center_x = annulus_center_x+ disc_r_shift*np.cos(disc_theta_shift)
  disc_center_y = annulus_center_y+ disc_r_shift*np.sin(disc_theta_shift)

  disc_points=sample_disc(n_disc, center=(disc_center_x,disc_center_y), radius=disc_radius)

  return [annulus_points, disc_points]

def sample_outside_annulus( disc_radius,annulus_inner_radius, annulus_outer_radius, n_points):

  n_annulus = math.ceil(n_points*((annulus_outer_radius)**2-
(annulus_inner_radius)**2)/((annulus_outer_radius)**2-(annulus_inner_radius)**2 + disc_radius**2))
  n_disc = n_points - n_annulus

  rng = np.random.default_rng()

  annulus_center_x = rng.uniform(-1.2,1.2)
  annulus_center_y = rng.uniform(-1.2,1.2)

  annulus_points=sample_annulus(n_annulus, center=(annulus_center_x,annulus_center_y),
r_inner=annulus_inner_radius, r_outer=annulus_outer_radius)

  disc_r_shift=rng.uniform(annulus_outer_radius+ disc_radius, annulus_outer_radius+ 4*disc_radius)
  disc_theta_shift=rng.uniform(0,2*np.pi)

  disc_center_x = annulus_center_x+ disc_r_shift*np.cos(disc_theta_shift)
  disc_center_y = annulus_center_y+ disc_r_shift*np.sin(disc_theta_shift)

  disc_points=sample_disc(n_disc, center=(disc_center_x,disc_center_y), radius=disc_radius)

  return [annulus_points, disc_points]
```

In [3]:
```
# @title
#Sampling Methods for 3d data



# ----------------------------
# Random rotation (uniform SO(3))
# ----------------------------
def random_rotation_matrix():
    u1, u2, u3 = np.random.rand(3)

    q1 = np.sqrt(1 - u1) * np.sin(2 * np.pi * u2)
    q2 = np.sqrt(1 - u1) * np.cos(2 * np.pi * u2)
    q3 = np.sqrt(u1) * np.sin(2 * np.pi * u3)
    q4 = np.sqrt(u1) * np.cos(2 * np.pi * u3)

    # Quaternion to rotation matrix
    R = np.array([
        [1 - 2*(q3**2 + q4**2), 2*(q2*q3 - q1*q4),     2*(q2*q4 + q1*q3)],
        [2*(q2*q3 + q1*q4),     1 - 2*(q2**2 + q4**2), 2*(q3*q4 - q1*q2)],
        [2*(q2*q4 - q1*q3),     2*(q3*q4 + q1*q2),     1 - 2*(q2**2 + q3**2)]
    ])
    return R


# ----------------------------
# Sample ellipsoid
# ----------------------------
def sample_ellipsoid(n, radii):
    """
    radii = (a, b, c)
    """
    x = np.random.normal(size=(n, 3))
    x /= np.linalg.norm(x, axis=1, keepdims=True)
    x *= np.random.rand(n, 1) ** (1/3)  # uniform in volume
    return x * np.array(radii)


# ----------------------------
# Sample torus
# ----------------------------
def sample_torus(n, R, r):
    """
    R = major radius
    r = minor radius
    """
    theta = 2 * np.pi * np.random.rand(n)
    phi = 2 * np.pi * np.random.rand(n)

    x = (R + r * np.cos(phi)) * np.cos(theta)
    y = (R + r * np.cos(phi)) * np.sin(theta)
    z = r * np.sin(phi)

    return np.column_stack([x, y, z])


# ----------------------------
# Main sampling function
# ----------------------------
def sample_wrapped_classes(
    n_ellipse=500,
    n_torus=500,
    ellipsoid_radii=(1.0, 0.6, 0.4),
    torus_radius=0.9,
    torus_thickness=0.15,
    shift_ellipsoid=False
):
    """
    Returns:
        X : (n_total, 3) array of points
        y : (n_total,) class labels (0 = ellipse, 1 = torus)
    """

    # Sample ellipse
    ellipse_pts = sample_ellipsoid(n_ellipse, ellipsoid_radii)

    # Choose torus radius so it surrounds the ellipse
    max_cross_section = max(ellipsoid_radii[0], ellipsoid_radii[1], ellipsoid_radii[2])
    torus_R =  torus_radius
    torus_r = torus_thickness

    torus_pts = sample_torus(n_torus, torus_R, torus_r)

    # Randomly rotate BOTH together
    R = random_rotation_matrix()
    ellipse_pts = ellipse_pts @ R.T
    R = random_rotation_matrix()
    torus_pts = torus_pts @ R.T

    if shift_ellipsoid==True:
      axis = R @ np.array([0.0, 0.0, 1.0])
      ellipse_pts=ellipse_pts+axis*max(ellipsoid_radii[0], ellipsoid_radii[1], ellipsoid_radii[2])

    return ellipse_pts, torus_pts
```

Now let's sample points for our SVM experiment. Mixup barcodes are calcluated on an ordered pair of sets of data points. Mixup Barcodes track the change in lifespans of homological features based on whether or not persistent homology computations are done with or without one of the classes of data.
Mixup Barcodes take the form of collections of triples of the form (b, pd, d). b is the birth of a homological feature when 1-parameter persistence computations are done on the the first data set, d is the death of that feature, also only considering the first class of data. pd is the ``pre-death'' of that feature. It is the death of this feature if we ran persistent homology computations on the union of the two data sets.

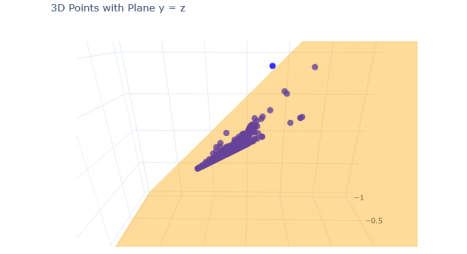
In [16]:
```
def sample_demo_torus(R, r, n_points, center=(0,0,0), rotation_matrix=None):
    """
    Samples points on a torus.
    """
    u = np.random.uniform(0, 2 * np.pi, n_points)
    v = np.random.uniform(0, 2 * np.pi, n_points)

    x = (R + r * np.cos(v)) * np.cos(u)
    y = (R + r * np.cos(v)) * np.sin(u)
    z = r * np.sin(v)

    points = np.vstack((x, y, z))
```

```
    if rotation_matrix is not None:
        points = rotation_matrix @ points

    points = points + np.array(center).reshape(3, 1)
    return points.T

def get_rotation_matrix(axis, theta):
    """
    Returns a 3x3 rotation matrix for rotation about a given axis by theta radians.
    """
    axis = np.asarray(axis)
    axis = axis / np.linalg.norm(axis)
    a = np.cos(theta / 2)
    b, c, d = -axis * np.sin(theta / 2)
    rm = np.array([
        [a*a + b*b - c*c - d*d, 2*(b*c - a*d),     2*(b*d + a*c)],
        [2*(b*c + a*d),         a*a + c*c - b*b - d*d, 2*(c*d - a*b)],
        [2*(b*d - a*c),         2*(c*d + a*b),     a*a + d*d - b*b - c*c]
    ])
    return rm

# Parameters
n_points = 500
R = 0.5    # Major radius
r = 0.25    # Minor radius

# First torus (left lobe of double torus)
points1a = sample_demo_torus(R=R, r=r, n_points=n_points, center=(-R - 0.2, 0, 0))

# Second torus (right lobe of double torus)
points1b = sample_demo_torus(R=R, r=r, n_points=n_points, center=(R + 0.2, 0, 0))

# Combine into one "double torus"
points1 = np.vstack((points1a, points1b))

# Second torus (red): rotated to link with the left torus
rotation = get_rotation_matrix(axis=[0, 1, 0], theta=np.pi / 2)  # rotate 90° around y-axis
points2 = sample_demo_torus(R=0.4, r=0.2, n_points=n_points, center=(-R - 0.2, R, 0),
rotation_matrix=rotation)

# Plotting
fig = go.Figure()

fig.add_trace(go.Scatter3d(
    x=points1[:, 0], y=points1[:, 1], z=points1[:, 2],
    mode='markers',
    marker=dict(size=2, color='blue', opacity=0.7),
    name='Double Torus (Blue)'
))

fig.add_trace(go.Scatter3d(
    x=points2[:, 0], y=points2[:, 1], z=points2[:, 2],
    mode='markers',
    marker=dict(size=2, color='red', opacity=0.7),
    name='Linked Torus (Red)'
))

fig.update_layout(
    title='Interactive Interlinked Tori in ℝ³',
    scene=dict(
        xaxis_title='X',
        yaxis_title='Y',
        zaxis_title='Z',
        aspectmode='data'
    ),
    width=800,
    height=700,
    showlegend=True
)

fig.show()
```



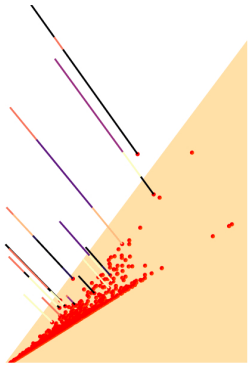
In [22]:
```
#Now let's calculate the mixup barcode
demo_mixup=Calculate_Mixup(points1, points2)

#We normalize this mixup barcode to fit into the unit cube in R^3
max_demo=max([arr[2] for arr in demo_mixup])

normed_mixup=max_demo**(-1)*demo_mixup
```

In [23]:
```
#We visualize these mixup barcodes as collections of points in R^3.  Note that we have plotted the plane
x_2=x_3. Points on this line have ``no mixup''; meaning these feature die at the same time whether we
include the second class of data points or not.
plot_mixup(normed_mixup)
```



In [25]:
```
#Now we may use MPMvect to map this mixup barcode to a vector. The method takes this entire collection of
points and maps it to a single list of real numbers, the length of which is dependent on user choice
(making a trade off between accuracy and computation time).
#To visualize this vector, we break it up into a collection of vectors, one for each point of the mixup
barcode. In the below plot, each point of the barcode has a series of color coded line segments erupting
form it. These colored line segments have lengths corresponding to the values in the vectorization of that
point.
#For example, if a point (considered as a mixup barcode on its own) was mapped to the vector [2,1,0.5,0] by
MPMvect, then we would see three, differently colored line segments erupting from the point, with lengths
equal to 2,1,and 0.5 respectively.
#Points with no mixup (pre-death=death) are mapped to the 0 vector in our method. This is a choice and we
could make it such that the only points that are mapped to the 0 vector are those such that b=pd=d.

n_list, p_list=collect_vertices_mixup(4)



#For visualization, we will individually vectorize each point. This is not the same as the MPMvect method
of vectorizing full barcodes. This is purely for visualization.
vectorizations=[None for _ in range(len(normed_mixup))]

for i,point in enumerate(normed_mixup):
  vector=vectorize_fast(n_list, p_list, np.array([point]), [1])
  vectorizations[i]=vector
```

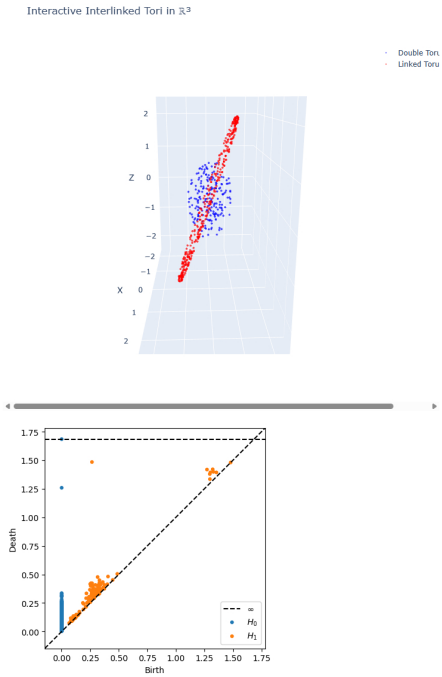
```
plot_mixup_nailbed(normed_mixup, vectorizations)
```

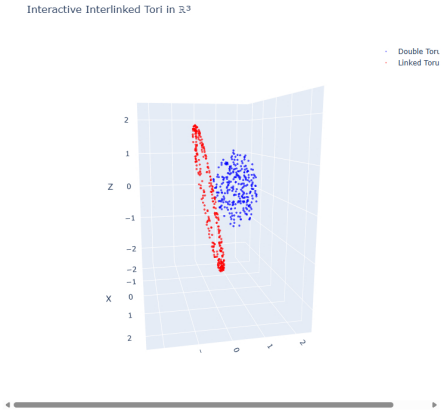


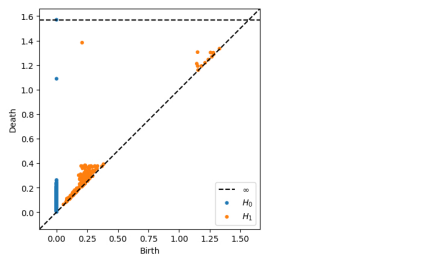
In [5]:

```
#We will randomize the generation of these pairs of data setls slightly for our SVM classification task

def rand_mix_pnts(n_points):

  rng = np.random.default_rng()

  er1 = rng.uniform(0.7,1)
  er2= rng.uniform(0.7,1)
  er3= rng.uniform(0.7,1)
  ellipsoid_radii=(er1, er2, er3)

  #n_ellipse=math.ceil(0.75*n_points*er1*er2*er3)
 # n_torus=n_points-n_ellipse


  torus_radius=rng.uniform(2*max(er1,er2,er3), 2.5*max(er1,er2,er3))
  torus_thickness=rng.uniform(0.05,0.06)

  n_torus=math.ceil(5*n_points*torus_thickness*(torus_radius))
  n_ellipse=n_points-n_torus



  points1, points2=sample_wrapped_classes(n_ellipse=n_ellipse, n_torus=n_torus,
ellipsoid_radii=ellipsoid_radii, torus_radius=torus_radius, torus_thickness=torus_thickness,
shift_ellipsoid=False)

  return points1, points2



def rand_nomix_pnts(n_points):


  rng = np.random.default_rng()

  er1 = rng.uniform(0.7,1)
  er2= rng.uniform(0.7,1)
  er3= rng.uniform(0.7,1)
  ellipsoid_radii=(er1, er2, er3)

  #n_ellipse=math.ceil(0.75*n_points*er1*er2*er3)
  #n_torus=n_points-n_ellipse

  torus_radius=rng.uniform(1.7*max(er1,er2,er3), 2.2*max(er1,er2,er3))
  torus_thickness=rng.uniform(0.05,0.06)

  n_torus=math.ceil(5*n_points*torus_thickness*(torus_radius))
  n_ellipse=n_points-n_torus

  points1, points2=sample_wrapped_classes(n_ellipse=n_ellipse, n_torus=n_torus,
ellipsoid_radii=ellipsoid_radii, torus_radius=torus_radius, torus_thickness=torus_thickness,
shift_ellipsoid=True)

  return points1, points2

#Let's generate such a pair of data sets, and calcluate the 1-parameter persistence diagram on the union of
the two data sets.
n_points=600
points1, points2=rand_mix_pnts(n_points)

fig = go.Figure()

fig.add_trace(go.Scatter3d(
    x=points1[:, 0], y=points1[:, 1], z=points1[:, 2],
    mode='markers',
    marker=dict(size=2, color='blue', opacity=0.7),
    name='Double Torus (Blue)'
))

fig.add_trace(go.Scatter3d(
    x=points2[:, 0], y=points2[:, 1], z=points2[:, 2],
    mode='markers',
    marker=dict(size=2, color='red', opacity=0.7),
    name='Linked Torus (Red)'
))

fig.update_layout(
    title='Interactive Interlinked Tori in ℝ³',
    scene=dict(
        xaxis_title='X',
        yaxis_title='Y',
        zaxis_title='Z',
        aspectmode='data'
    ),
    width=800,
    height=700,
    showlegend=True
)

xmin=ymin=zmin=-2.5
xmax=ymax=zmax=2.5
fig.update_layout(
    scene=dict(
        xaxis=dict(range=[xmin, xmax]),
        yaxis=dict(range=[ymin, ymax]),
        zaxis=dict(range=[zmin, zmax]),
        aspectmode='data'
    )
)

fig.show()


# Compute persistence diagram using Vietoris-Rips filtration
points_3d = np.concatenate((points1, points2))
diagrams = ripser(points_3d)['dgms']

# Plot persistence diagram
plot_diagrams(diagrams, show=True)



#Now let's repeat this sampling, but with the ellipsoid shifted mostly outside the torus.

n_points=600
points1, points2=rand_nomix_pnts(n_points)

fig = go.Figure()

fig.add_trace(go.Scatter3d(
    x=points1[:, 0], y=points1[:, 1], z=points1[:, 2],
    mode='markers',
    marker=dict(size=2, color='blue', opacity=0.7),
    name='Double Torus (Blue)'
))

fig.add_trace(go.Scatter3d(
    x=points2[:, 0], y=points2[:, 1], z=points2[:, 2],
    mode='markers',
    marker=dict(size=2, color='red', opacity=0.7),
    name='Linked Torus (Red)'
))

fig.update_layout(
    title='Interactive Interlinked Tori in ℝ³',
    scene=dict(
        xaxis_title='X',
        yaxis_title='Y',
        zaxis_title='Z',
        aspectmode='data'
    ),
    width=800,
    height=700,
    showlegend=True
)
xmin=ymin=zmin=-2.5
xmax=ymax=zmax=2.5
```

```
fig.update_layout(
    scene=dict(
        xaxis=dict(range=[xmin, xmax]),
        yaxis=dict(range=[ymin, ymax]),
        zaxis=dict(range=[zmin, zmax]),
        aspectmode='data'
    )
)

fig.show()

# Compute persistence diagram using Vietoris-Rips filtration
points_3d = np.concatenate((points1, points2))
diagrams = ripser(points_3d)['dgms']

# Plot persistence diagram
plot_diagrams(diagrams, show=True)
```







In [6]:
```
#Defining a single function to calculate persistence landscape for 1-parameter persistence


def compute_landscapes(points):
  alpha = gd.AlphaComplex(points=points)
  st = alpha.create_simplex_tree()

  st.persistence()


  H1 = st.persistence_intervals_in_dimension(1)
  keys = H1[:, 0] - H1[:, 1]
  A_sorted = H1[np.argsort(keys)]
  #print(A_sorted[0])
  landscape = Landscape(resolution=500, num_landscapes=5)
  L = landscape.fit_transform([H1])[0]
  return  H1[:,0].min(), H1[:,1].max(), L
```

In [7]:
```
#We now repeat this sampling 50 times for each case

Class_Inside=[]
Class_Outside=[]
iterations=50

for i in range(iterations):

  points1, points2=rand_mix_pnts(n_points)
  Class_Inside.append([points1, points2])

  points1, points2=rand_nomix_pnts(n_points)
  Class_Outside.append([points1, points2])
```

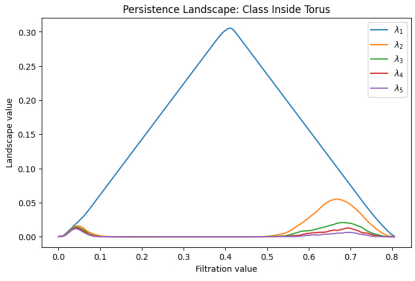
In [8]:
```
#Generating Landscapes
Inside_Annulus_Landscape=[]
```

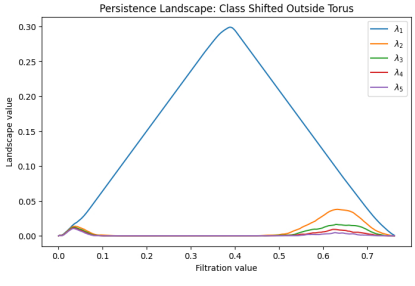
```
Outside_Annulus_Landscapes=[]

Inside_bounds=[]
Outside_bounds=[]
#default_landscape_size = 500
for i in range(iterations):
  X=np.concatenate((Class_Inside[i][0], Class_Inside[i][1]))
  Y=np.concatenate((Class_Outside[i][0], Class_Outside[i][1]))

  min_x, max_x, Inside_Annulus_Landscape = compute_landscapes(X)
  min_y, max_y, Outside_Annulus_Landscape = compute_landscapes(Y)

  Inside_Annulus_Landscapes.append(Inside_Annulus_Landscape)
  Outside_Annulus_Landscapes.append(Outside_Annulus_Landscape)

  Inside_bounds.append([min_x, max_x])
  Outside_bounds.append([min_y, max_y])

  sys.stdout.write(f"\rCompleted {i+1}/{iterations}")
  sys.stdout.flush()
print()
```

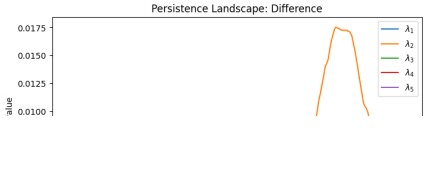
```
Completed 50/50
```

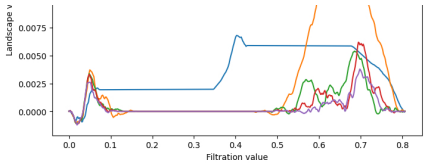
In [9]:
```
#Let's plot the average landscapes and the difference between them
L_Ann_stack = np.vstack(Inside_Annulus_Landscapes)   # shape: (50, D)
L_Ann_avg_flat = L_Ann_stack.mean(axis=0)      # shape: (D,)
L_Ann_avg = L_Ann_avg_flat.reshape(5, 500)

# Manual domain
bmin = min([v[0] for v in Inside_bounds])
dmax = max([v[1] for v in Inside_bounds])
t = np.linspace(bmin, dmax, 500)

plt.figure(figsize=(8,5))
for k in range(5):
    plt.plot(t, L_Ann_avg[k], label=f"$\lambda_{k+1}$")

plt.xlabel("Filtration value")
plt.ylabel("Landscape value")
plt.title("Persistence Landscape: Class Inside Torus")
plt.legend()
plt.show()



L_Dis_stack = np.vstack(Outside_Annulus_Landscapes)
L_Dis_avg_flat = L_Dis_stack.mean(axis=0)      # shape: (D,)
L_Dis_avg = L_Dis_avg_flat.reshape(5, 500)


bmin = min([v[0] for v in Outside_bounds])
dmax = max([v[1] for v in Outside_bounds])
t = np.linspace(bmin, dmax, 500)

plt.figure(figsize=(8,5))
for k in range(5):
    plt.plot(t, L_Dis_avg[k], label=f"$\lambda_{k+1}$")

plt.xlabel("Filtration value")
plt.ylabel("Landscape value")
plt.title("Persistence Landscape: Class Shifted Outside Torus")
plt.legend()
plt.show()

#Let's look at the difference of the two
L_dif=L_Ann_avg-L_Dis_avg
bmin = min(min([v[0] for v in Inside_bounds]), min([v[0] for v in Outside_bounds]))
dmax = max(max([v[1] for v in Inside_bounds]), max([v[1] for v in Outside_bounds]))
t = np.linspace(bmin, dmax, 500)

plt.figure(figsize=(8,5))
for k in range(5):
    plt.plot(t, L_dif[k], label=f"$\lambda_{k+1}$")

plt.xlabel("Filtration value")
plt.ylabel("Landscape value")
plt.title("Persistence Landscape: Difference")
plt.legend()
plt.show()
```

```
<>:13: SyntaxWarning:

invalid escape sequence '\l'

<>:34: SyntaxWarning:

invalid escape sequence '\l'

<>:50: SyntaxWarning:

invalid escape sequence '\l'

<>:13: SyntaxWarning:

invalid escape sequence '\l'

<>:34: SyntaxWarning:

invalid escape sequence '\l'

<>:50: SyntaxWarning:

invalid escape sequence '\l'

/tmp/ipython-input-1763631104.py:13: SyntaxWarning:

invalid escape sequence '\l'

/tmp/ipython-input-1763631104.py:34: SyntaxWarning:

invalid escape sequence '\l'

/tmp/ipython-input-1763631104.py:50: SyntaxWarning:

invalid escape sequence '\l'
```



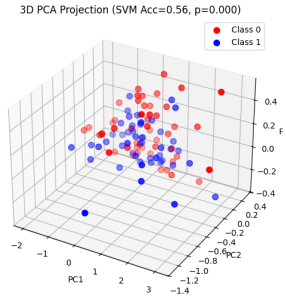






In [10]:
```
#We now attempt classification using Support Vector Machine. We also calculate a p-value and plot the 3D
PCA projection
do_SVM_and_PCA(Inside_Annulus_Landscapes, Outside_Annulus_Landscapes)
```

```
0.56
0.0
```



Now let's try MPHvect on Mixup Barcodes with these classes of data

In [11]:
```
#First we calculate mixup barcodes, which tracks the change in lifespans of homological features when the
class of the data points in the ellipsoid is included in the set of data points of the torus.
#These features take the form of collections of triples of the form (b, pd, d). b is the birth of a
homological feature in the data set contained in the torus. d is the death of that feature (in the standard
1-parameter persistence sesnse).
#pd is the ``pre-death'' of that feature. It is the death of this feature if we ran the simplicial complex
filtration on the union of the data from the torus AND the ellipsoid.
#calculating mixup takes about 1 minute, 30 seconds per iteration

Inside_Mixup=[]
Outside_Mixup=[]
for i in range(iterations):
  inside_disc=Class_Inside[i][0]
  inside_ann=Class_Inside[i][1]
  in_mixup = Calculate_Mixup(inside_ann, inside_disc)

  outside_disc=Class_Outside[i][0]
  outside_ann=Class_Outside[i][1]
  out_mixup = Calculate_Mixup(outside_ann, outside_disc)

  Inside_Mixup.append(in_mixup)
  Outside_Mixup.append(out_mixup)
  sys.stdout.write(f"\rCompleted {i+1}/{iterations}")
  sys.stdout.flush()
print()
```

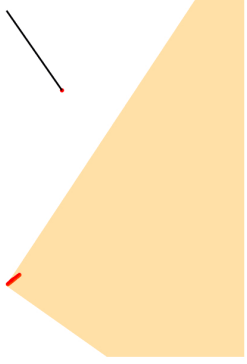
```
Completed 50/50
```

In [12]:
```
#normalize mixup barcodes to fit into unit cube
max_inside=max([arr[2] for arr in Inside_Mixup[0]])
max_outside=max([arr[2] for arr in Outside_Mixup[0]])
my_max=max(max_inside, max_outside)

Normed_Inside_Mixup=np.array([None for _ in range(iterations)])
Normed_Outside_Mixup=np.array([None for _ in range(iterations)])

for i in range(iterations):
  in_mixup=Inside_Mixup[i]
  out_mixup=Outside_Mixup[i]

  normed_inside=my_max**(-1)*in_mixup
  normed_outside=my_max**(-1)*out_mixup

  Normed_Inside_Mixup[i]=normed_inside
  Normed_Outside_Mixup[i]=normed_outside


print(max([arr[2] for arr in Normed_Inside_Mixup[0]]))
```

```
1.0
```

Let's visualize one of these mixup barcodes as points in R^3. Note that we have plotted the plane x_2=x_3. Points on this line have ``no mixup'': meaning these feature die at the same time whether we include the ellipsoid into our data set or not.

In [13]:
```
#set up vectorization

n_list, p_list=collect_vertices_mixup(4)

#pick the mixup barcode to visualize
first_mixup_barcode=Normed_Inside_Mixup[0]

#For visualization, we will individually vectorize each point. This is not the same as the MPHvect method
of vectorizing full barcodes. This is purely for visualization.
vectorizations=[None for _ in range(len(first_mixup_barcode))]

for i,point in enumerate(first_mixup_barcode):
  vector=vectorize_fast(n_list, p_list, np.array([point]), [1])
  vectorizations[i]=vector


plot_mixup_nailbed(first_mixup_barcode, vectorizations)

#In this example, there is one significant feature that has a large difference between death and pre-death.
This is represented as the single poitn off the plane x_2=x_3. The colored line segments erupting from this
point, have lengths corresponding to the values in the vectorization of that point.
#For example, if a point was mapped to the vector [2,1,0.5,0], then we would see three, differently colored
line segments erupting from the point, with lengths equal to 2,1,and 0.5 respectively.
#Points with no mixup (pre-death=death) are mapped to the 0 vector in our method. This is a choice and we
could make it such that the only points that are mapped to the 0 vector are those such that b=pd=d.
```



Now let's vectorize all the mixup barcodes. This converts each mixup barcode to a list of real numbers.

In [14]:
```
#set up vectorization. The collect_vertices_mixup function determines which functionals in the Schauder
Basis are used. (In theory we use all, but of course we can only compute finitely many).
# The input of this function is the number of refinements of the Coxeter Freudenthal Kuhn triangulation
that are taken. We use all functionals centered at vertices of these refinements that are contained in the
unit square.

n_list, p_list=collect_vertices_mixup(4)
```

```
Inside_Vectors=[]
Outside_Vectors=[]


for i in range(iterations):
  in_mixup=Normed_Inside_Mixup[i]
  out_mixup=Normed_Outside_Mixup[i]

  mults=[1 for _ in range(len(in_mixup))]
  in_vec = vectorize_fast(n_list, p_list, in_mixup, mults)

  mults=[1 for _ in range(len(out_mixup))]
  out_vec = vectorize_fast(n_list, p_list, out_mixup, mults)
  Inside_Vectors.append(in_vec)
  Outside_Vectors.append(out_vec)

  sys.stdout.write(f"\rCompleted {i+1}/{iterations}")
  sys.stdout.flush()
print()
```

```
Completed 50/50
```

In [15]:
```
do_SVM_and_PCA(Inside_Vectors, Outside_Vectors)
```

```
0.93
0.0
```