Short exercises Lecture 9

Exercise 1 (Exercise 1 Lecture 5)**.** Use your favorite software to draw a persistence diagram and landscape functions for the Vietoris–Rips persistence module of the following point cloud.

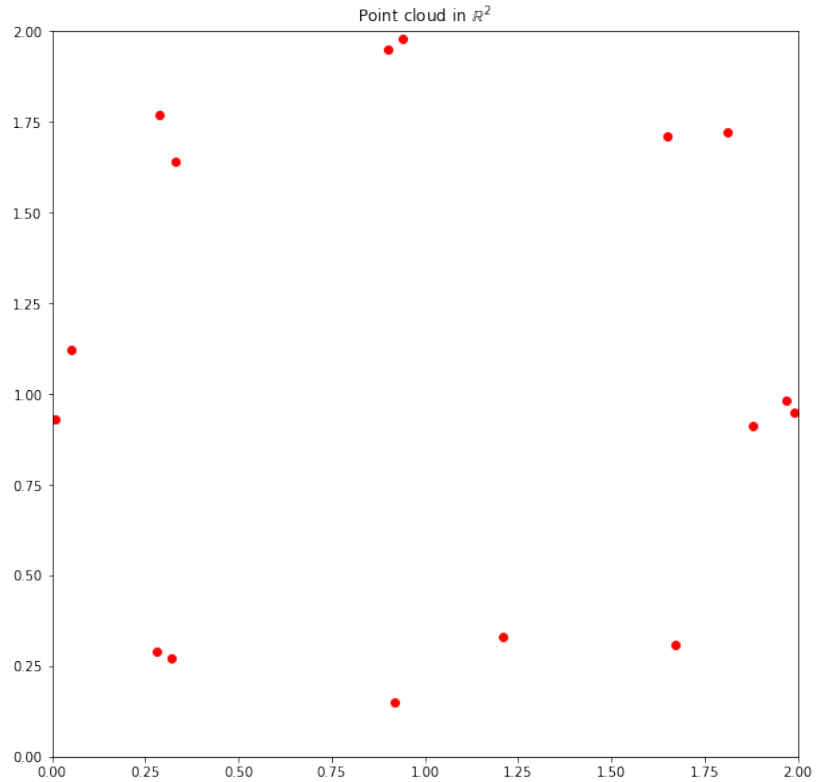
$$(0.01, 0.93)\ (0.05, 1.12)\ (0.28, 0.29)\ (0.29, 1.77)$$
$$(0.32, 0.27)\ (0.33, 1.64)\ (0.90, 1.95)\ (0.92, 0.15)$$
$$(0.94, 1.98)\ (1.21, 0.33)\ (1.65, 1.71)\ (1.67, 0.31)$$
$$(1.81, 1.72)\ (1.88, 0.91)\ (1.97, 0.98)\ (1.99, 0.95)$$

Solution. We are going to use the `GUDHI` library in order to compute the Vietoris-Rips filtration and to compute the persistence diagram. However, the `Python` version of `GUDHI` does not implement landscape functions (but the `C++` version does). In this lecture's short exercise, we coded a piece of software in `Mathematica` to compute the landscape functions. We are going to use it for this purpose. Let us start by letting X be the point cloud above. We can plot it right away with `Python`, using the `matplotlib` library:

```
import numpy as np
import matplotlib.pyplot as plt

point_cloud = [[0.01, 0.93], [0.05, 1.12], [0.28, 0.29], [0.29, 1.77], [0.32, 0.27],
               [0.33, 1.64], [0.90, 1.95], [0.92, 0.15], [0.94, 1.98], [1.21, 0.33],
               [1.65, 1.71], [1.67, 0.31], [1.81, 1.72], [1.88, 0.91], [1.97, 0.98],
               [1.99, 0.95]]
point_cloud_mat = np.matrix(point_cloud)
plt.figure(figsize=(10,10))
plt.plot(point_cloud_mat[:,0], point_cloud_mat[:,1], 'ro')
plt.xlim([0,2])
plt.ylim([0,2])
plt.title(r'Point cloud in $\mathbb{R}^2$')
plt.show()
```

This yields:



Let us compute now the Vietoris-Rips filtration of X, $R_\varepsilon(X)$, using `GUDHI`:

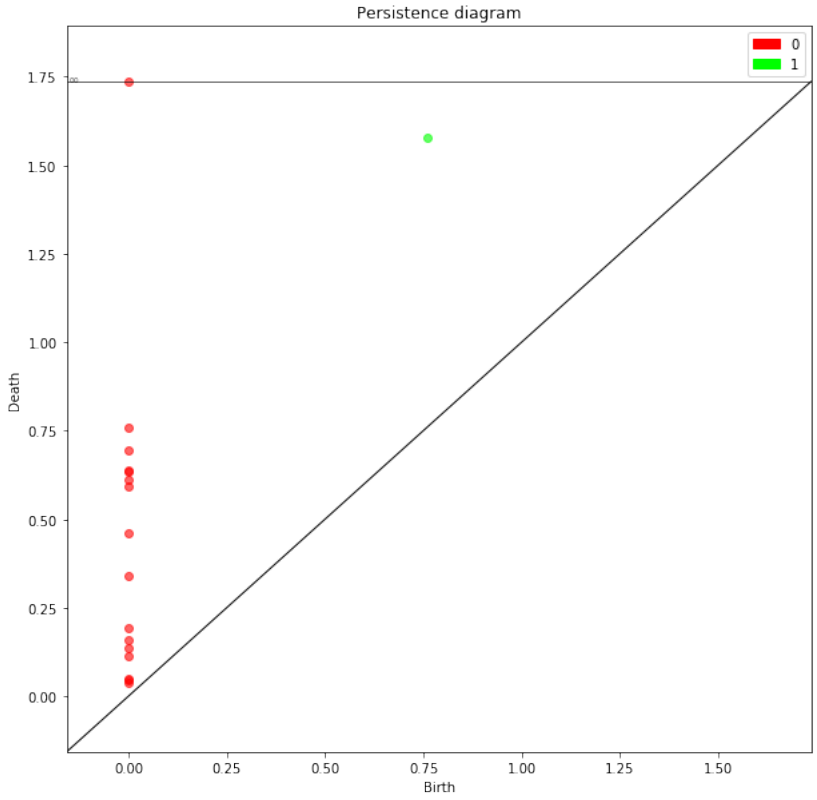
```
import gudhi as gd

rips = gd.RipsComplex(points=point_cloud)
simplex_tree = rips.create_simplex_tree(max_dimension=len(point_cloud))
```

The second line just states that we want to compute the Vietoris-Rips complex of X, and the third line does the actual computation. The parameter `max_dimension` tells the program up until which simplex dimension do we want it to compute. That is, if we set it to k, it computes the k-skeleton of $R_\varepsilon(X)$. This is a small example and we can just compute up to the maximum dimension (we know that $R_\infty(X) = \Delta^{|X|}$).
Let us now compute the persistence diagram:

```
diag = simplex_tree.persistence(min_persistence=0.01)
plt.figure(figsize=(10,10))
plot = gd.plot_persistence_diagram(persistence=diag, legend=True)
plot.show()
```

The first line computes the birth and death values for the persistence. In fact, `diag` is a list such that, each entry on the list is a tuple with 2 values: the first one being the degree of the homology and the second one the birth and death values for that.
The argument in first line (`min_persistence`) tells `GUDHI` that it should not consider the points that have a life-span less than that value. That is, the points (b, d) such that $d - b <$ `min_persistence`.
Here is what the snippet above outputs:



As we explained above, `Python`'s wrapper of `GUDHI` does not implement landscapes (in fact, it would be an interesting Sunday's afternoon exercise to do it). So, we are going to use `Mathematica` for this task. Why `Mathematica`? Well, just for the convenience of defining functions symbolically.
But before we even start up the software, we must get the input values for the function. For this, we are going to use the following code:

```
print('{', end='')
for x in diag:
    if x[1][1] == float('inf'): continue
    print("{\%.4lf, \%.4lf}," \% (x[1][0], x[1][1]), end=' ')
print('}')
```

This outputs the input for our `Mathematica` code, which is the following:

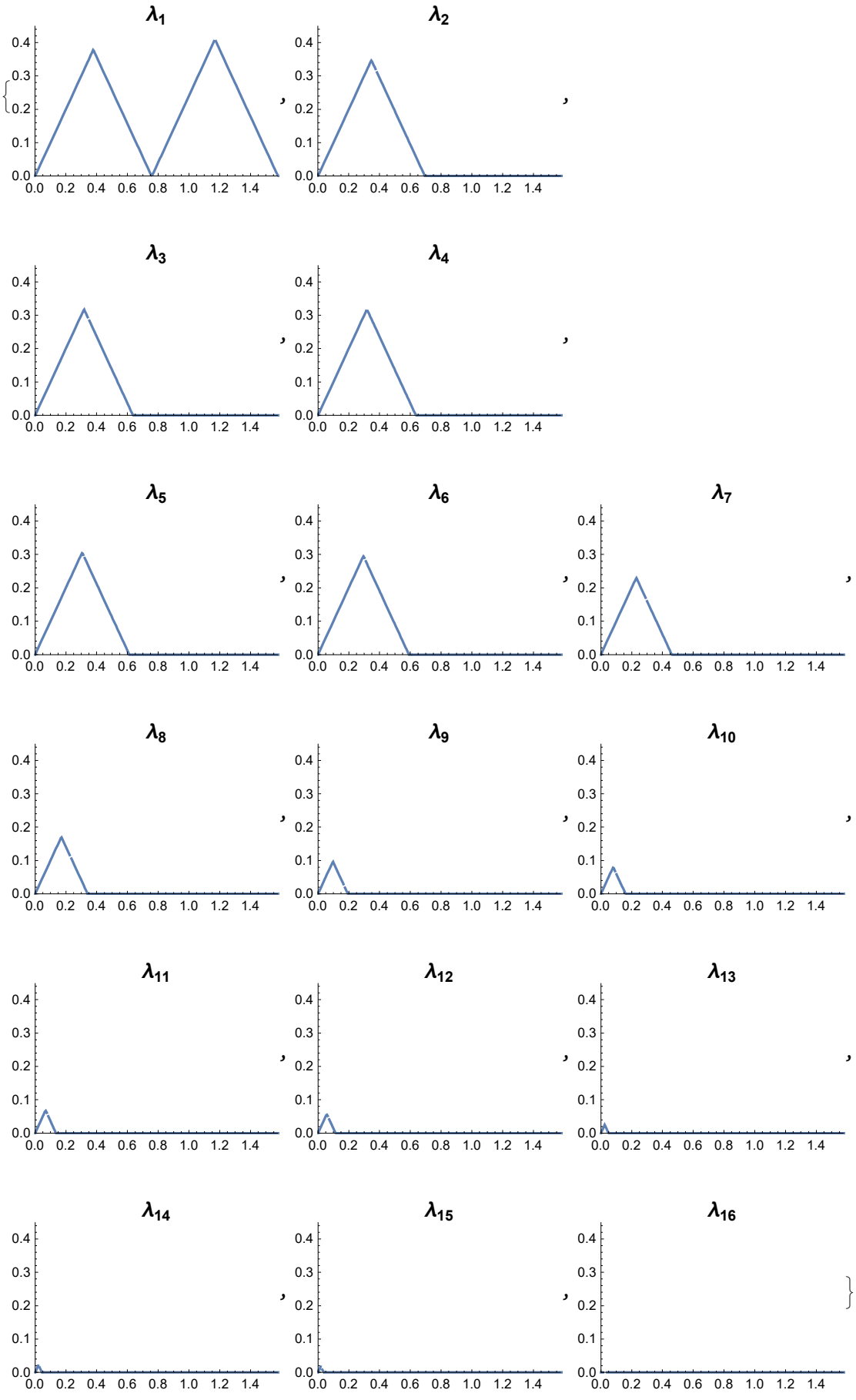
```
In[1]:= Λ[t_, b_, d_] := Max[{0, Min[{t - b, d - t}]}]
        λ[t_, k_, bd_] := Module[{l = {}, i = 1},
          For[i = 1, i ≤ Length[bd], i++, AppendTo[l, Λ[t, bd[[i]][[1]], bd[[i]][[2]]]]];
          Return[Sort[l, (#1 > #2) &][[k]]]]

In[3]:= bd = {{0.7596, 1.5781}, {0.0000, 0.7571}, {0.0000, 0.6946}, {0.0000, 0.6360},
          {0.0000, 0.6357}, {0.0000, 0.6119}, {0.0000, 0.5906}, {0.0000, 0.4604},
          {0.0000, 0.3413}, {0.0000, 0.1942}, {0.0000, 0.1603}, {0.0000, 0.1360},
          {0.0000, 0.1140}, {0.0000, 0.0500}, {0.0000, 0.0447}, {0.0000, 0.0361}}
Out[3]= {{0.7596, 1.5781}, {0., 0.7571}, {0., 0.6946}, {0., 0.636}, {0., 0.6357},
         {0., 0.6119}, {0., 0.5906}, {0., 0.4604}, {0., 0.3413}, {0., 0.1942},
         {0., 0.1603}, {0., 0.136}, {0., 0.114}, {0., 0.05}, {0., 0.0447}, {0., 0.0361}}

In[4]:= Table[Plot[λ[k, i, bd], {k, 0, Length[bd]},
          PlotRange → {{Min[bd], Max[bd]}, {0, 0.45}}, ImageSize → Small,
          PlotLabel → Style[StringForm["λ``", i], FontSize → 14, FontWeight → Bold]], {i,
          1, Length[bd]}]
```

This outputs the following figures:

Out[4]=

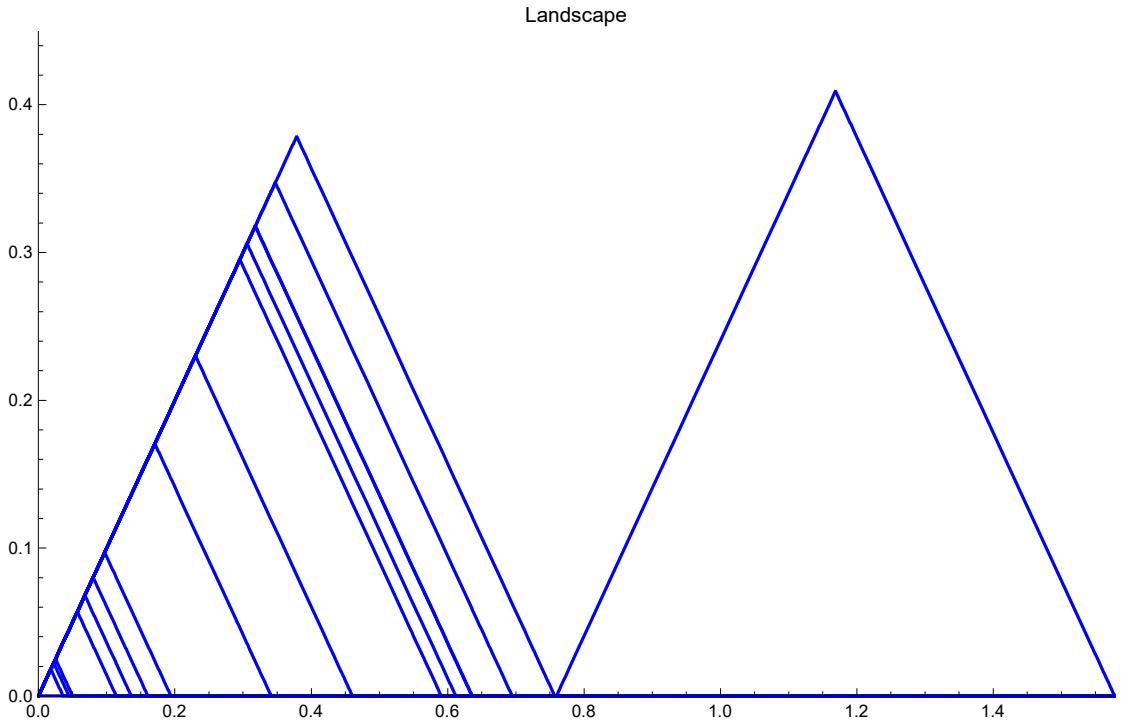


Finally, the whole landscape is the following:

```
In[5]:= functions = Table[λ[k, i, bd], {i, 1, Length[bd]}];
        legends = Table[StringForm["λ``", i], {i, 1, Length[bd]}];

In[7]:= r = Plot[functions, {k, Min[bd], Max[bd]},
          PlotRange → {{Min[bd], Max[bd]}, {0, 0.45}},
          PlotStyle → Blue, ImageSize → Large, PlotLabel → "Landscape"]
```

Out[7]=



□

Exercise 2 (Exercise 2 Lecture 6)**.** Draw a barcode for H_0, H_1 and H_2, and landscape functions for H_1 of the Vietoris-Rips persistence module of the following point cloud in $\mathbb{R}^3$:

$$(1.05, 2.00, 2.16)\ (2.96, 1.84, 2.09)\ (2.49, 2.77, 1.87)\ (1.99, 2.42, 2.82)$$
$$(1.59, 1.42, 2.66)\ (2.27, 1.59, 1.16)\ (1.69, 1.24, 2.60)\ (2.22, 2.76, 2.53)$$
$$(2.50, 2.74, 2.07)\ (2.55, 1.59, 1.30)\ (1.80, 1.15, 1.47)\ (1.31, 2.40, 1.44)$$
$$(1.92, 1.06, 1.64)\ (2.29, 1.97, 1.01)\ (2.25, 2.94, 2.29)\ (2.38, 2.57, 2.71)$$
$$(1.19, 1.83, 1.55)\ (1.53, 2.80, 2.24)\ (1.52, 2.82, 1.81)\ (1.66, 2.58, 2.66)$$
$$(2.05, 1.76, 2.98)\ (1.41, 1.96, 1.29)\ (1.70, 2.51, 2.81)\ (1.79, 1.74, 2.88)$$
$$(2.89, 1.84, 2.16)\ (2.16, 1.63, 1.08)\ (1.62, 1.27, 2.63)\ (2.91, 2.27, 2.29)$$
$$(1.05, 2.05, 1.59)\ (2.60, 1.91, 2.82)\ (1.21, 1.68, 2.64)\ (1.67, 3.00, 1.80)$$
$$(1.76, 1.09, 2.21)\ (0.99, 1.66, 2.29)\ (2.57, 1.95, 1.17)\ (1.31, 1.45, 2.63)$$
$$(0.99, 1.90, 1.84)\ (2.88, 2.63, 2.05)\ (2.13, 1.96, 1.08)\ (2.62, 2.76, 2.47)$$
$$(2.35, 2.64, 2.69)\ (2.52, 2.94, 2.19)\ (1.68, 1.25, 2.55)\ (1.03, 2.11, 1.74)$$
$$(2.49, 2.15, 2.89)\ (1.37, 2.46, 1.35)\ (2.21, 1.05, 2.40)\ (1.20, 1.73, 2.36)$$
$$(1.67, 1.15, 1.95)\ (2.45, 2.27, 1.14)$$

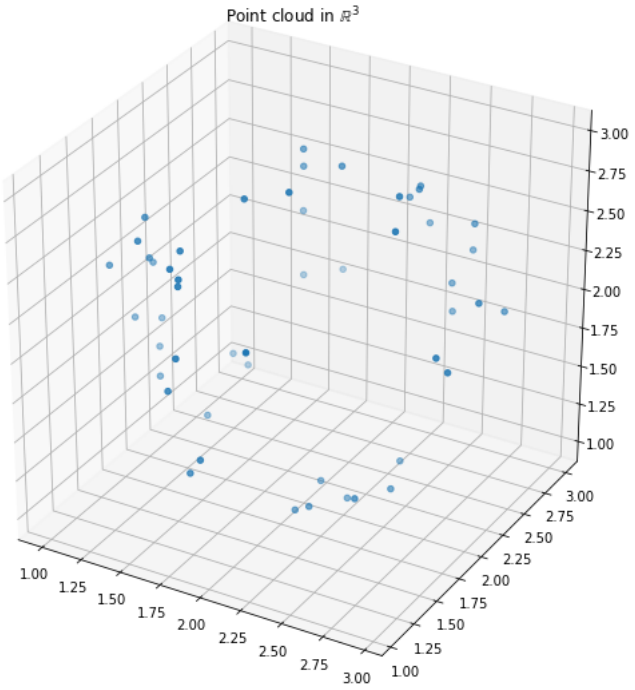
Solution. As with the previous exercise, we are going to use `GUDHI` to plot the barcodes and `Mathematica` for the landscape functions.
First, let us visualize the dataset:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

point_cloud = [[1.05, 2.00, 2.16], [2.96, 1.84, 2.09], [2.49, 2.77, 1.87], [1.99, 2.42, 2.82],
[1.59, 1.42, 2.66], [2.27, 1.59, 1.16], [1.69, 1.24, 2.60], [2.22, 2.76, 2.53],
[2.50, 2.74, 2.07], [2.55, 1.59, 1.30], [1.80, 1.15, 1.47], [1.31, 2.40, 1.44],
[1.92, 1.06, 1.64], [2.29, 1.97, 1.01], [2.25, 2.94, 2.29], [2.38, 2.57, 2.71],
[1.19, 1.83, 1.55], [1.53, 2.80, 2.24], [1.52, 2.82, 1.81], [1.66, 2.58, 2.66],
[2.05, 1.76, 2.98], [1.41, 1.96, 1.29], [1.70, 2.51, 2.81], [1.79, 1.74, 2.88],
[2.89, 1.84, 2.16], [2.16, 1.63, 1.08], [1.62, 1.27, 2.63], [2.91, 2.27, 2.29],
[1.05, 2.05, 1.59], [2.60, 1.91, 2.82], [1.21, 1.68, 2.64], [1.67, 3.00, 1.80],
[1.76, 1.09, 2.21], [0.99, 1.66, 2.29], [2.57, 1.95, 1.17], [1.31, 1.45, 2.63],
[0.99, 1.90, 1.84], [2.88, 2.63, 2.05], [2.13, 1.96, 1.08], [2.62, 2.76, 2.47],
[2.35, 2.64, 2.69], [2.52, 2.94, 2.19], [1.68, 1.25, 2.55], [1.03, 2.11, 1.74],
[2.49, 2.15, 2.89], [1.37, 2.46, 1.35], [2.21, 1.05, 2.40], [1.20, 1.73, 2.36],
[1.67, 1.15, 1.95], [2.45, 2.27, 1.14]]
point_cloud_mat = np.matrix(point_cloud)

fig = plt.figure(figsize=(10,10))
ax = plt.axes(projection='3d')
ax.scatter3D(point_cloud_mat[:,0], point_cloud_mat[:,1], point_cloud_mat[:,2])
plt.title(r'Point cloud in $\mathbb{R}^3$')
plt.show()
```

This outputs the following image:

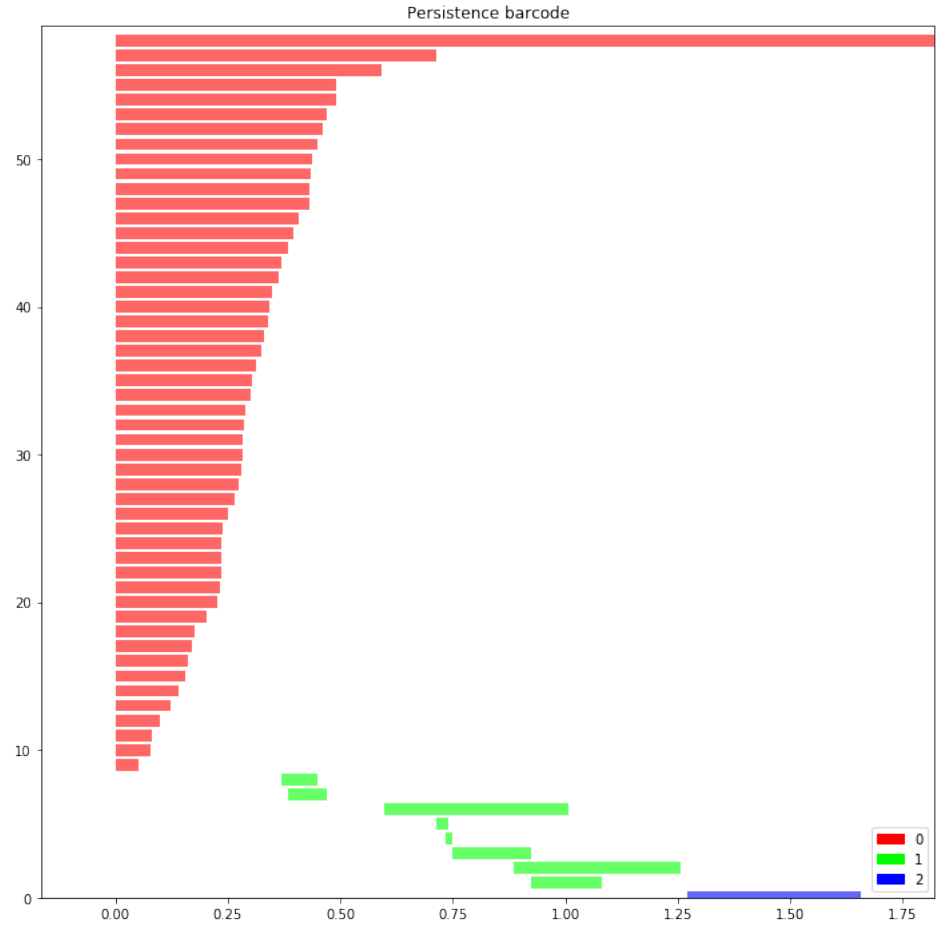


Now we compute the barcode using `GUDHI`:

```
rips = gd.RipsComplex(points=point_cloud)
max_dim = 2
simplex_tree = rips.create_simplex_tree(max_dimension=max_dim+1)

diag = simplex_tree.persistence(min_persistence=0.01)
plt.figure(figsize=(12,12))
plot = gd.plot_persistence_barcode(diag, legend=True)
plot.show()
```

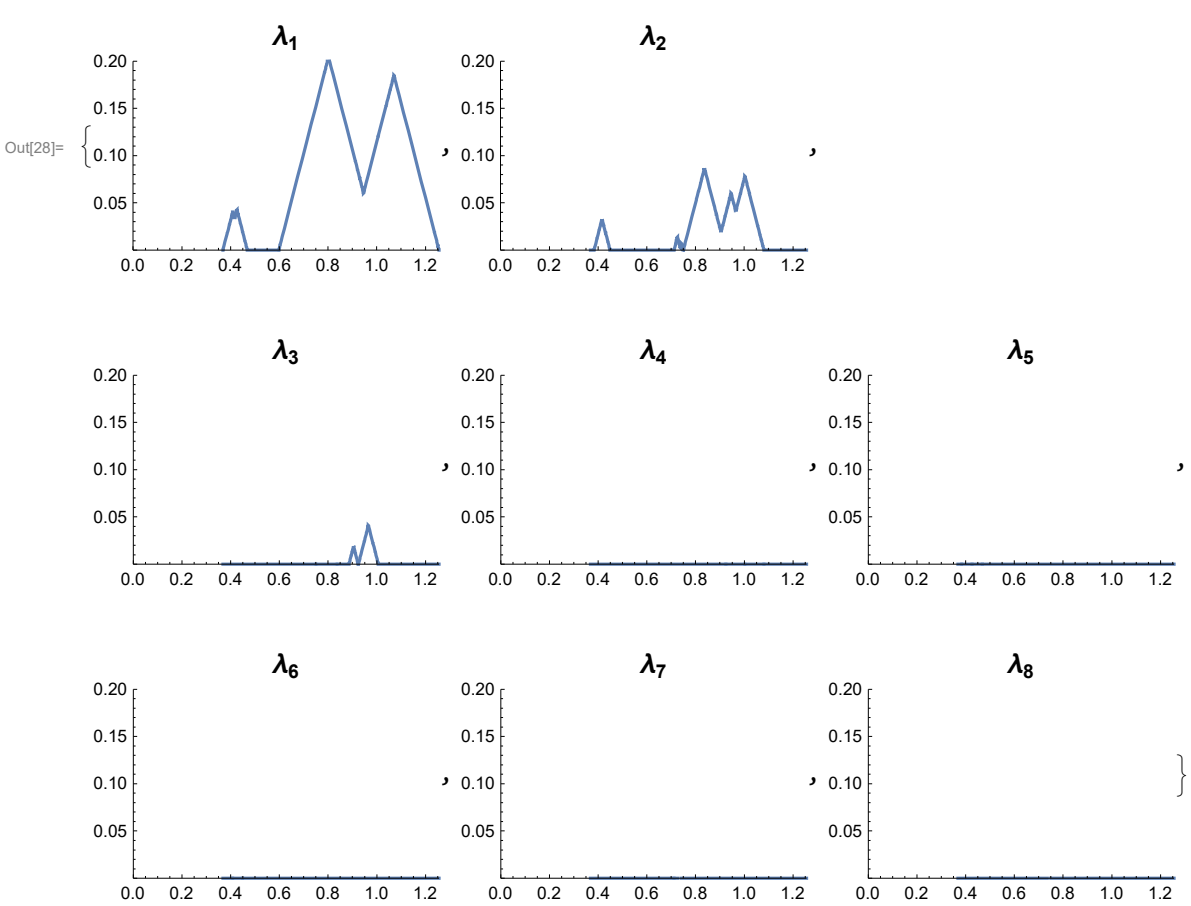
This outputs:



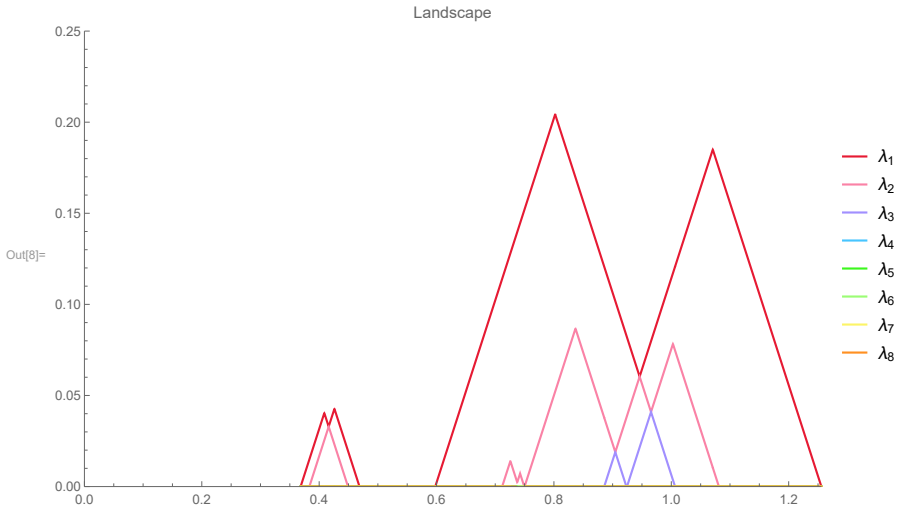
We have to extract only the H_1 points to produce the landscape function. For this, we use a similar code from the one before:

```
print('{', end='')
for x in diag:
    if x[0] != 1: continue #This selects only H_1 points
    if x[1][1] == float('inf'): continue
    print("{\%.4lf, \%.4lf}," \% (x[1][0], x[1][1]), end=' ')
print('}')
```

Finally, `Mathematica` (using the same code as before) outputs the following plots landscape functions:



Note that $\lambda_k = 0$ for $k \geq 4$. All together is:



□