Assignment5

November 6, 2024

0.0.1 CS2101 - Programming for Science and Finance

Prof. Götz Pfeiffer School of Mathematical and Statistical Sciences University of Galway

---

1 Computer Lab 5

---

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```python
rng = np.random.default_rng()
```

1.1 1. Sorting and Ranking

1. Use the random number generator to make a list of 10 random values between 0 and 1, perhaps as a numpy array `values` of length 10.

[3]:
```python
values = rng.random(10)
values
```

[3]:
```
array([0.22994784, 0.31439973, 0.62037894, 0.76771509, 0.64708903,
       0.79624752, 0.64031441, 0.49329014, 0.21683796, 0.80375952])
```

2. Look up the documentation for the functions `np.sort` and `np.argsort`. What do they have in common, what is the difference? Use `np.sort` to compute as array `ranked` a sorted copy of the `values` array. Use `np.argsort` to compute as array `ranks` the list of sorting indices for the array `values`.

`np.sort` : returns a sorted copy of an array `np.argsort` : returns the indexes that would sort the array

[4]:
```python
ranked = np.sort(values)
ranks = np.argsort(values)

display(ranked)
display(ranks)
```

```
array([0.21683796, 0.22994784, 0.31439973, 0.49329014, 0.62037894,
       0.64031441, 0.64708903, 0.76771509, 0.79624752, 0.80375952])
```

```
array([8, 0, 1, 7, 2, 6, 4, 3, 5, 9])
```

3. Check that the array `values[ranks]` is equal to array `ranked`.

[5]:
```
np.array_equal(values[ranks], ranked)
```

[5]:
```
True
```

1.2 2. Random Points

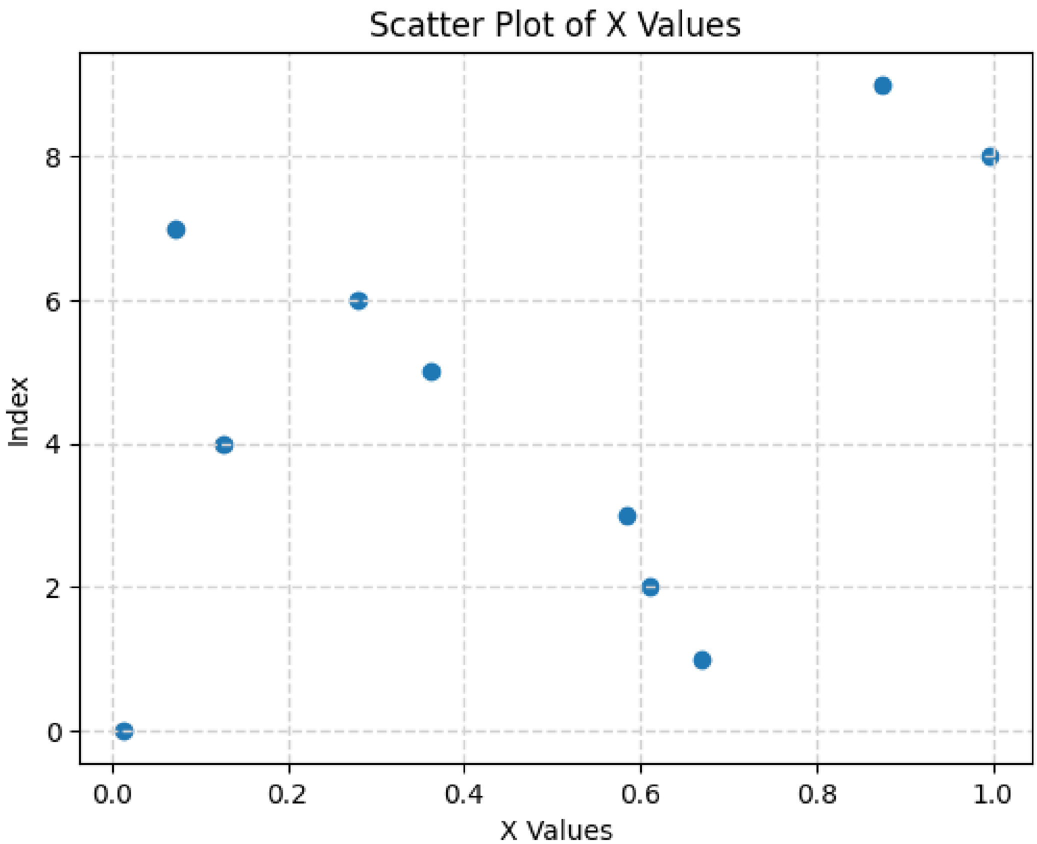
1. Use the random number generator to make a list of 10 random points in the x, y-plane, with coordinates between 0 and 1, perhaps as a numpy array `points` of shape 10×2.

[6]:
```
points = rng.random((10,2))
points
```

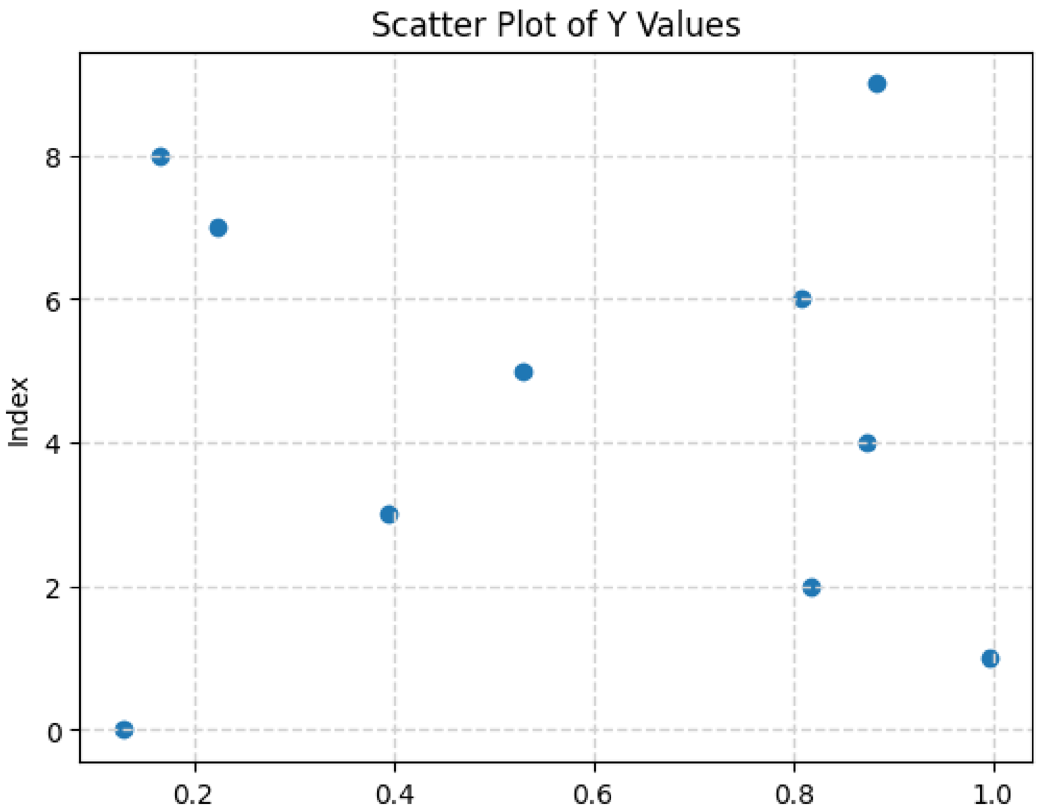
[6]:
```
array([[0.01286391, 0.12914863],
       [0.67046658, 0.99604695],
       [0.6101219 , 0.81781967],
       [0.58412274, 0.39470653],
       [0.12604319, 0.87288211],
       [0.362144  , 0.53037739],
       [0.27840896, 0.80844223],
       [0.07298052, 0.22352523],
       [0.9956572 , 0.16587742],
       [0.87461011, 0.88242586]])
```

2. Use slicing and indices to select the x-values of your points and plot them against `range(10)`. Do the same for the y-values of your points.

[7]:
```
plt.scatter(points[:,0], np.arange(10))
plt.grid(True, color='lightgray', linestyle='--', linewidth=1)
plt.xlabel('X Values')
plt.ylabel('Index')
plt.title('Scatter Plot of X Values')
plt.show();
```

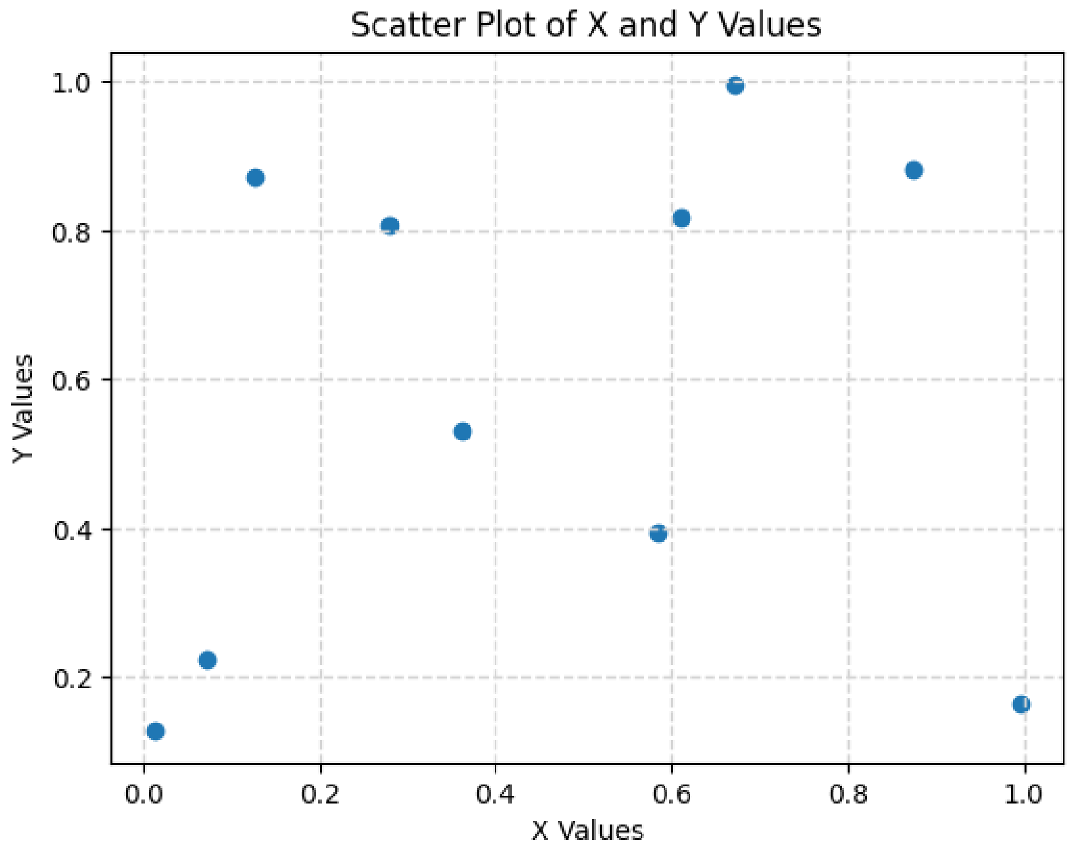


[8]:
```
plt.scatter(points[:,1], np.arange(10))
plt.grid(True, color='lightgray', linestyle='--', linewidth=1)
plt.ylabel('Index')
plt.title('Scatter Plot of Y Values')
plt.show();
```



3. Plot the 10 points as a scatter plot in the x, y-plane.

[9]:
```
plt.scatter(points[:,0], points[:,1])
plt.grid(True, color='lightgray', linestyle='--', linewidth=1)
plt.xlabel('X Values')
plt.ylabel('Y Values')
plt.title('Scatter Plot of X and Y Values')
plt.show();
```



1.3 3. Extra Dimensions

- One way in which Numpy extends Python's indexing scheme to its multidimensional arrays is by allowing us to add an extra dimension. This is done by using the constant `None`, or `np.newindex`, as an index-or-slice. The effect is the same as if using an extra argument `1` in a `reshape` command. Look up the documentation for `np.newindex`.

1. What is the shape of `points[:,np.newaxis,:]` ?

[10]:
```
points[:,np.newaxis,:].shape
```

[10]: (10, 1, 2)

2. What is the shape of `points[np.newaxis,:,:]` ?

[11]:
```
points[np.newaxis,:,:].shape
```

[11]: (1, 10, 2)

3. What is the shape of `points[None,:,None,:]`?

[12]:
```
points[None, :, None, :].shape
```

[12]: (1, 10, 1, 2)

1.4 4. Broadcasting

- Recall Broadcasting: two numpy arrays can be added if the have the "same" shape. Same shape means same number of dimensions and in each dimension, the same size unless one of the sizes is 1. In that case the entry is repeated as often as needed in that dimension. Lokok up the documentation of `np.broadcast` and `np.boradcast_to`.

1. Compute `np.broadcast_to(points[:,None,:], (10,10,2))` and assign the result to `points1`. What is the shape of `points1`?

[13]:
```
points1 = np.broadcast_to(points[:,None,:], (10,10,2))
points1.shape
```

[13]: (10, 10, 2)

2. Compute `np.broadcast_to(points[None,:,:], (10,10,2))` and assign the result to `points2`. What is the shape of `points2`?

[14]:
```
points2 = np.broadcast_to(points[None,:,:], (10,10,2))
points2.shape
```

[14]: (10, 10, 2)

3. Compute the sum of `points1` and `points2` and assign it to `sums`. What are the entries in the resulting array?

[15]:
```
sums = points1 + points2
sums
```

[15]:
```
array([[[0.02572783, 0.25829725],
        [0.68333049, 1.12519558],
        [0.62298582, 0.94696829],
        [0.59698665, 0.52385516],
        [0.13890711, 1.00203074],
        [0.37500792, 0.65952602],
        [0.29127288, 0.93759086],
        [0.08584443, 0.35267385],
        [1.00852111, 0.29502605],
        [0.88747402, 1.01157448]],

       [[0.68333049, 1.12519558],
        [1.34093315, 1.9920939 ],
        [1.28058848, 1.81386662],
        [1.25458932, 1.39075348],
        [0.79650977, 1.86892906],
        [1.03261058, 1.52642434],
        [0.94887554, 1.80448918],
```

```
 [0.7434471 , 1.21957218],
 [1.66612378, 1.16192437],
 [1.54507668, 1.87847281]],

[[0.62298582, 0.94696829],
 [1.28058848, 1.81386662],
 [1.22024381, 1.63563933],
 [1.19424464, 1.2125262 ],
 [0.7361651 , 1.69070178],
 [0.97226591, 1.34819706],
 [0.88853087, 1.6262619 ],
 [0.68310242, 1.0413449 ],
 [1.6057791 , 0.98369709],
 [1.48473201, 1.70024552]],

[[0.59698665, 0.52385516],
 [1.25458932, 1.39075348],
 [1.19424464, 1.2125262 ],
 [1.16824548, 0.78941306],
 [0.71016593, 1.26758864],
 [0.94626674, 0.92508392],
 [0.8625317 , 1.20314876],
 [0.65710326, 0.61823176],
 [1.57977994, 0.56058395],
 [1.45873285, 1.27713239]],

[[0.13890711, 1.00203074],
 [0.79650977, 1.86892906],
 [0.7361651 , 1.69070178],
 [0.71016593, 1.26758864],
 [0.25208639, 1.74576423],
 [0.4881872 , 1.40325951],
 [0.40445216, 1.68132434],
 [0.19902372, 1.09640734],
 [1.12170039, 1.03875953],
 [1.0006533 , 1.75530797]],

[[0.37500792, 0.65952602],
 [1.03261058, 1.52642434],
 [0.97226591, 1.34819706],
 [0.94626674, 0.92508392],
 [0.4881872 , 1.40325951],
 [0.72428801, 1.06075479],
 [0.64055297, 1.33881962],
 [0.43512453, 0.75390262],
 [1.3578012 , 0.69625481],
 [1.23675411, 1.41280325]],
```

```
[[0.29127288, 0.93759086],
 [0.94887554, 1.80448918],
 [0.88853087, 1.6262619 ],
 [0.8625317 , 1.20314876],
 [0.40445216, 1.68132434],
 [0.64055297, 1.33881962],
 [0.55681793, 1.61688446],
 [0.35138949, 1.03196746],
 [1.27406616, 0.97431965],
 [1.15301907, 1.69086809]],

[[0.08584443, 0.35267385],
 [0.7434471 , 1.21957218],
 [0.68310242, 1.0413449 ],
 [0.65710326, 0.61823176],
 [0.19902372, 1.09640734],
 [0.43512453, 0.75390262],
 [0.35138949, 1.03196746],
 [0.14596104, 0.44705046],
 [1.06863772, 0.38940265],
 [0.94759063, 1.10595108]],

[[1.00852111, 0.29502605],
 [1.66612378, 1.16192437],
 [1.6057791 , 0.98369709],
 [1.57977994, 0.56058395],
 [1.12170039, 1.03875953],
 [1.3578012 , 0.69625481],
 [1.27406616, 0.97431965],
 [1.06863772, 0.38940265],
 [1.9913144 , 0.33175484],
 [1.8702673 , 1.04830328]],

[[0.88747402, 1.01157448],
 [1.54507668, 1.87847281],
 [1.48473201, 1.70024552],
 [1.45873285, 1.27713239],
 [1.0006533 , 1.75530797],
 [1.23675411, 1.41280325],
 [1.15301907, 1.69086809],
 [0.94759063, 1.10595108],
 [1.8702673 , 1.04830328],
 [1.74922021, 1.76485171]]])
```

4. Check that `sums` is equal to the sum of `points[:,None]` and `points[None,:]`. Look up the documentation of `np.array_equal`.

[16]:
```python
np.array_equal(sums, (points[:,None] + points[None,:]))
```

[16]: True

1.5 5. Nearest Neighbors

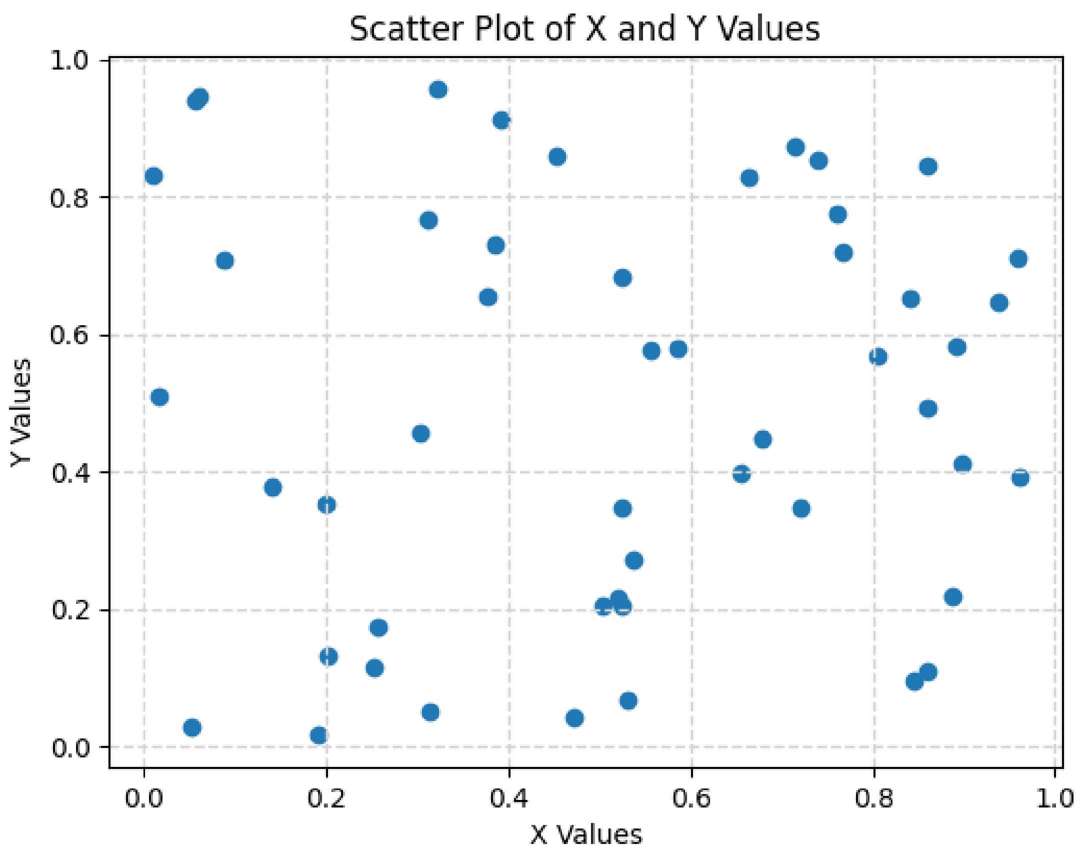
- Let's plot 50 random points in the x, y-plane, and connect each point with its two nearest neighbors by an edge, as follows.

1. Use the random number generator to make a list of 50 random points in the x, y-plane, with coordinates between 0 and 1, as a numpy array `points` of shape 50×2

[17]:
```python
points = rng.random((50,2))
```

2. Plot the 50 points as a scatter plot in the x, y-plane.

[18]:
```python
plt.scatter(points[:,0], points[:,1])
plt.grid(True, color='lightgray', linestyle='--', linewidth=1)
plt.xlabel('X Values')
plt.ylabel('Y Values')
plt.title('Scatter Plot of X and Y Values')
plt.show();
```



3. Using `None` as a slice, convert the array `points` into an array of shape `(50, 1, 2)` and assign the result to ‘points1

[19]:
```
points1 = points[:,None,:]
points1.shape
```

[19]: `(50, 1, 2)`

4. Using `None` as a slice, convert the array `points` into an array of shape `(1, 50, 2)` and assign the result to `points2`.

[20]:
```
points2 = points[None,:,:]
points2.shape
```

[20]: `(1, 50, 2)`

5. Now, compute and rank the distances between any pair of points as follows. Recall that the square distance between points $a = (a_0, a_1)$ and $b = (b_0, b_1)$ in the x, y-plane is the sum of the squared differences in each dimension:

$$\|a - b\|^2 = (a_0 - b_0)^2 + (a_1 - b_1)^2.$$

First, using two suitably reshaped (via `None` slices) variants of the `points` array, compute a 3D array `diffs` whose i, j-entry is `points[i] - points[j]`.

[21]:
```
diffs = points1 - points2
```

6. Compute the squares of the values in `diffs` and assign the result to `diffs2`.

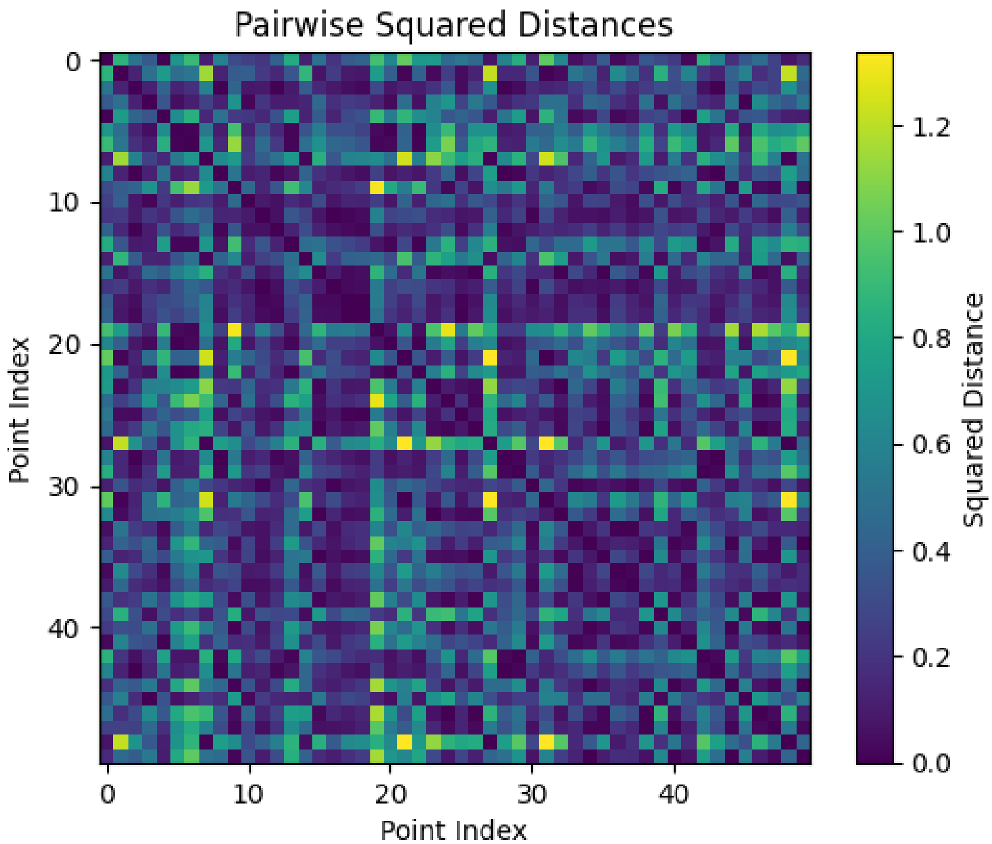
[22]:
```
diffs2 = np.square(diffs)
```

7. Using `np.sum` with argument `axis=-1`, add the x and y components of the squared differences in `diff2` and assign the result to `dist2`.

[23]:
```
dist2 = np.sum(diffs2, axis=-1)
```

8. Optionally, display `dist2` as an image

[24]:
```
plt.imshow(dist2)
plt.colorbar(label='Squared Distance')
plt.title('Pairwise Squared Distances')
plt.xlabel('Point Index')
plt.ylabel('Point Index')
plt.show()
```



9. Check if the diagonal is 0. Look up the documentation of `np.diagonal`.

[25]:
```
np.allclose(np.diagonal(dist2), 0)
```

[25]: True

10. Next, for each point a, rank its neighbours by distance. That is: apply `np.argsort` to each row of `dist2`. Assign the result to `nearest`.

[26]:
```
nearest = np.argsort(dist2)
```

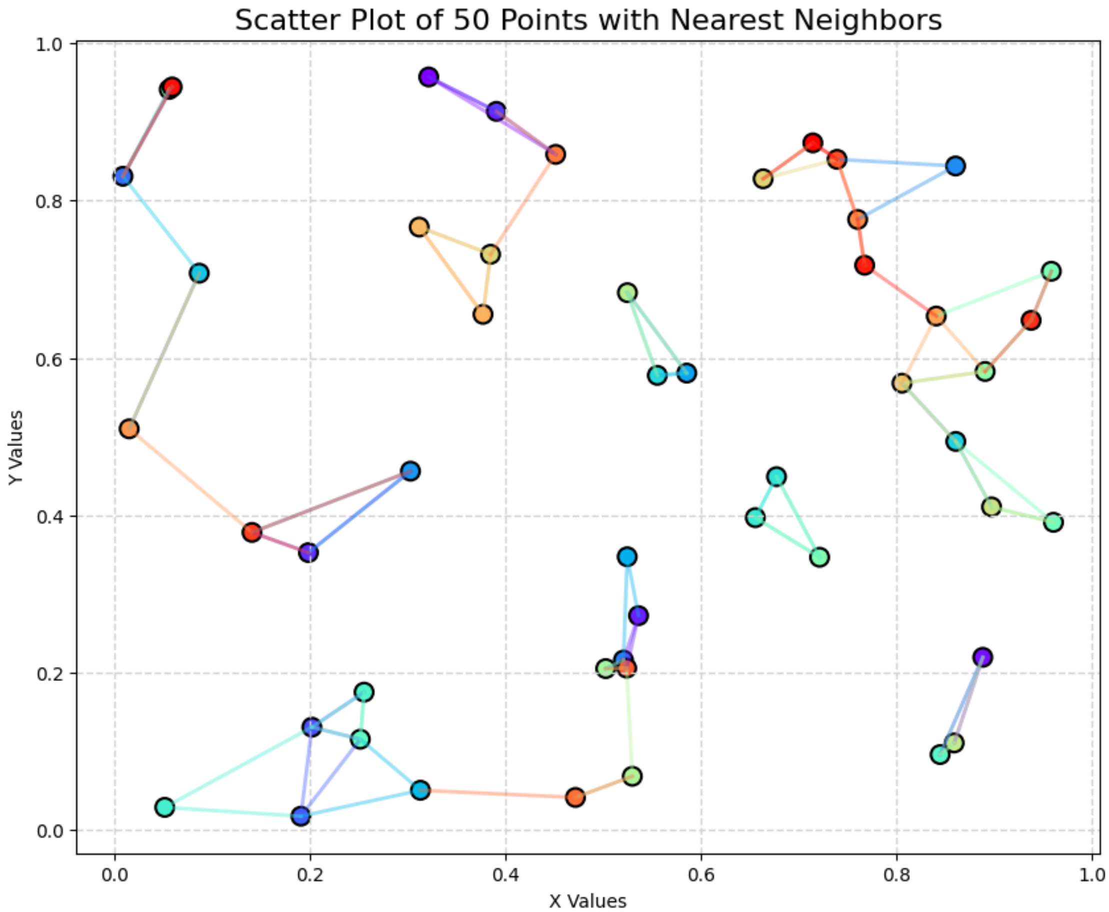
11. Finally, plot the points and join each point with its two nearest neighbors by an edge: start with a scatter plot of all 50 points. Then, loop over all the points. For each point `points[i]` find the indices `j` of its two nearest neighbors, then use a command like

```
plt.plot(*zip(points[i], points[j]), color='r', alpha=0.5)
```

to draw an edge between them.

[27]:
```
import matplotlib.cm as cm

# Define colors for points
```

```python
colors = cm.rainbow(np.linspace(0, 1, len(points)))

# Set up plot
fig, ax = plt.subplots(figsize=(10, 8))
ax.scatter(points[:, 0], points[:, 1], s=100, color=colors, edgecolor='k',␣
↪linewidth=1.5)

# Add grid lines
ax.grid(True, color='lightgray', linestyle='--', linewidth=1)

# Add labels and title
ax.set_xlabel('X Values')
ax.set_ylabel('Y Values')
ax.set_title('Scatter Plot of 50 Points with Nearest Neighbors', fontsize=16)

for i in range(len(points)):
    for j in nearest[i, 1:3]:  # Get the indices of the two nearest neighbors
        ax.plot(*zip(points[i], points[j]), color=colors[i], linewidth=2,␣
↪alpha=0.4) # Plot a line between the two points

plt.show()
```



Scatter Plot of 50 Points with Nearest Neighbors
Y Values
X Values
0.0
0.2
0.4
0.6
0.8
1.0