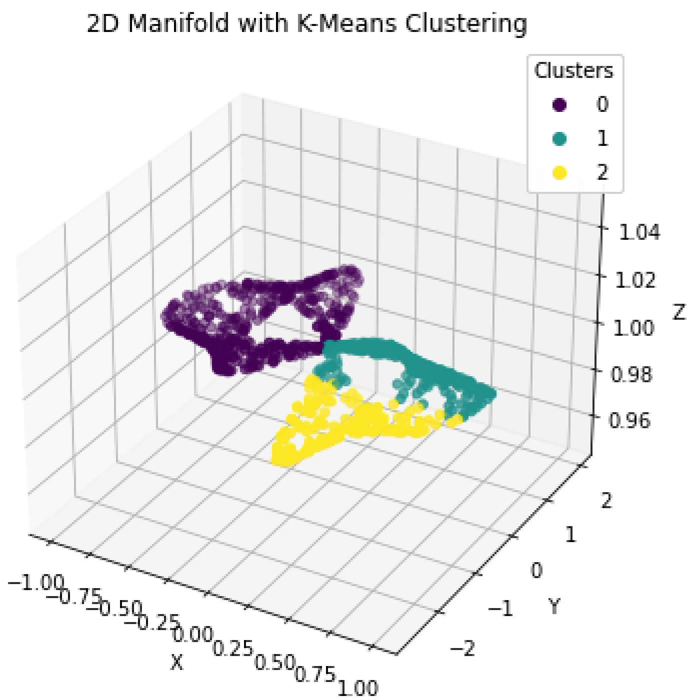
In [94]:

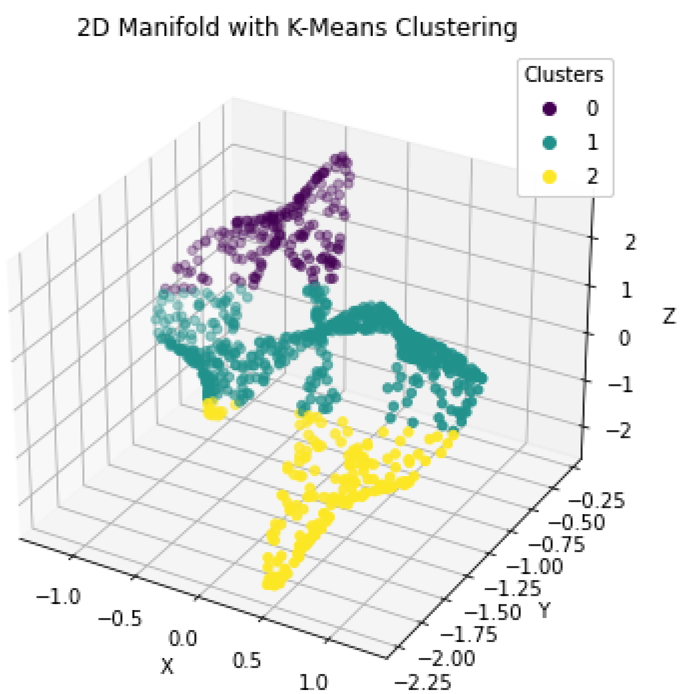
```
import torch
PI = torch.tensor(math.pi)
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
# Function to generate 3D sphere points
def generate_2d_manifold_points(num_points, radius=1.0):
    phi = torch.rand(num_points) * 2 * PI
    theta = torch.rand(num_points) * PI
    x = radius * torch.cos(phi)/(1+torch.sin(theta)**2)
    y = radius * phi * torch.sin(phi)* torch.cos(phi)/(1+torch.cos(theta)**2)
    z = radius * torch.cos(theta*0)
    points_3d = torch.stack([x, y, z], dim=1)
    return points_3d
# Generate 1000 points on a 3D sphere with a radius of 1.0
num_points = 1000
sphere_points = generate_2d_manifold_points(num_points)
# Perform k-means clustering on the points
num_clusters = 3 # Adjust the number of clusters as needed
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
sphere_points_np = StandardScaler().fit_transform(sphere_points.numpy()) #␣
#Standardize the data for k-means
labels = kmeans.fit_predict(sphere_points_np)
# Plot the 3D points using a scatter plot with coloring based on k-means clusters
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
# Scatter plot with colored points
scatter = ax.scatter(sphere_points[:, 0], sphere_points[:, 1], sphere_points[:,2], c=l
# Legend
legend = ax.legend(*scatter.legend_elements(), title="Clusters")
ax.add_artist(legend)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('2D Manifold with K-Means Clustering')
plt.show()
```



In [126…

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Generate or load your sphere_points data here

# Rotate the data 6 times in 120-degree increments
rotated_points = []
for i in range(6):
    # Calculate rotation angles
    theta = i * 2 * np.pi / 6  # Angle around z-axis
    phi = np.arccos(1/3)  # Angle around y-axis

    # Rotation matrices around x, y, and z axes
    Rx = np.array([[1, 0, 0],
                   [0, np.cos(phi), -np.sin(phi)],
                   [0, np.sin(phi), np.cos(phi)]])

    Ry = np.array([[np.cos(theta), 0, np.sin(theta)],
                   [0, 1, 0],
                   [-np.sin(theta), 0, np.cos(theta)]])

    R = Rx @ Ry  # Combined rotation matrix

    # Apply rotation to the data
    rotated_data = (R @ sphere_points.numpy().T).T
    rotated_points.append(rotated_data)

# Concatenate rotated points
rotated_points = np.concatenate(rotated_points, axis=0)

# Standardize the rotated data for k-means
rotated_points = StandardScaler().fit_transform(rotated_points)

# Perform k-means clustering
```

```
kmeans = KMeans(n_clusters=3, random_state=0)
labels = kmeans.fit_predict(rotated_points[0:1000,:])

# Plot the 3D points with clustering
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot with colored points
scatter = ax.scatter(rotated_points[0:1000, 0], rotated_points[0:1000, 1], rotated_poi

# Legend
legend = ax.legend(*scatter.legend_elements(), title="Clusters")
ax.add_artist(legend)

# Set labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('2D Manifold with K-Means Clustering')

plt.show()
```



In [127…
```
np.shape(rotated_points[0:1000,:])
```

Out[127]:
```
(1000, 3)
```

In [128…
```
Y=rotated_points[0:1000,:]
# License: Copyright (c) 2014, the Open Data Science Initiative
# License: https://www.elsevier.com/legal/elsevier-website-terms-and-conditions
URL = "https://raw.githubusercontent.com/sods/ods/master/datasets/guo_qpcr.csv"
import pandas as pd
#df = pd.read_csv(URL, index_col=0)
df=pd.DataFrame(Y)
print("Data shape: {}\n{}\n".format(df.shape, "-" * 21))
print("Show a small subset of the data:")
df.head()
```

```
Data shape: (1000, 3)
---------------------

Show a small subset of the data:
```

Out[128]:

	0	1	2
0	-0.093311	-1.134799	0.453088
1	-0.291215	-0.407745	2.319990
2	0.684050	-1.005301	0.785608
3	0.807584	-1.122017	0.485908
4	-0.394990	-1.220494	0.233043

In [129…

```
data = torch.tensor(df.values, dtype=torch.get_default_dtype())
# we need to transpose data to correct its shape
y = data.t()
```

In [122…

```
y.size(1)
# we setup the mean of our prior over X
X_prior_mean = torch.zeros(y.size(1), 3) # shape: 437 x
```

In [123…

```
from mpl_toolkits.axes_grid.inset_locator import inset_axes
```

In [15]:

```
import GPy
```

In [130…

```
n = 1000
m = 40
np.random.seed(1)
x = np.random.uniform(-1, 1, n)
c = np.digitize(x, np.linspace(-1,1,12))-1
cols = np.asarray(sns.color_palette('Accent',12))[c]
labels = kmeans.fit_predict(Y)

fig, axes = plt.subplots(2,3,tight_layout=True,figsize=(15,10))
axit = axes.flat
for lr in range(1):
    ax = next(axit)
    m = GPy.models.GPLVM(Y.copy(), 2)
    m.optimize(messages=1, gtol=0.0001, clear_after_finish=True)
    msi = m.get_most_significant_input_dimensions()[:2]

    is_ = m.kern.input_sensitivity().copy()
    is_ /= is_.max()

    YBGPLVM = m.X[:,msi] * is_[np.array(msi)]
    #m.kern.plot_ARD(ax=ax)
    ax.scatter(*YBGPLVM.T, c=labels, cmap='viridis', lw=0)
    ax.set_title('restart: ${}$'.format(lr))
    ax.set_xlabel('dimension ${}$'.format(msi[0]))
    ax.set_ylabel('dimension ${}$'.format(msi[1]))
    a = inset_axes(ax,
                    width="30%", # width = 30% of parent_bbox
                    height='20%', # height : 1 inch
                    loc=1)
    sns.barplot(x=np.array(msi), y=is_[np.array(msi)], label='input-sens', ax=a)
```

```
    a.set_title('sensitivity')
    a.set_xlabel('dimension')
```

```
Running L-BFGS-B (Scipy implementation) Code:
  runtime   i      f              |g|        
    00s00  0000   2.908709e+03           nan 
    02s44  0010  -2.084776e+04   1.680969e+10 
    09s45  0039  -2.366513e+04   1.621264e+09 
    28s46  0119  -2.384056e+04   7.476361e+06 
 01m23s51  0344  -2.394187e+04   5.163321e+07 
 03m46s01  0967  -2.416711e+04   4.532459e+06 
 03m54s10  1002  -2.417434e+04   5.489478e+06 
Runtime:  03m54s10
Optimization status: Maximum number of f evaluations reached
```

```
 C:\Python39\lib\site-packages\IPython\core\pylabtools.py:151: UserWarning:This figur
e includes Axes that are not compatible with tight_layout, so results might be incorr
ect.
```