```
#In this project we will be implementing the stochastic block model which can be used to p
#eigenvectors of any given function
#we will be implementing our own custom stochastic block model algorithm capable of using
# to ease Signal Processing on Graphs and plotting any given given array like parameter (i
#Note that the pygsp module is used only for plotting the eigenvectors and eigenvalues gra
#we will then start by importing the necesary libraries to help in the implementation
```

```
pip install pygsp  #installing the pygsp module for graph plotting
```

```
Requirement already satisfied: pygsp in /usr/local/lib/python3.7/dist-packages (0.5.
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
```

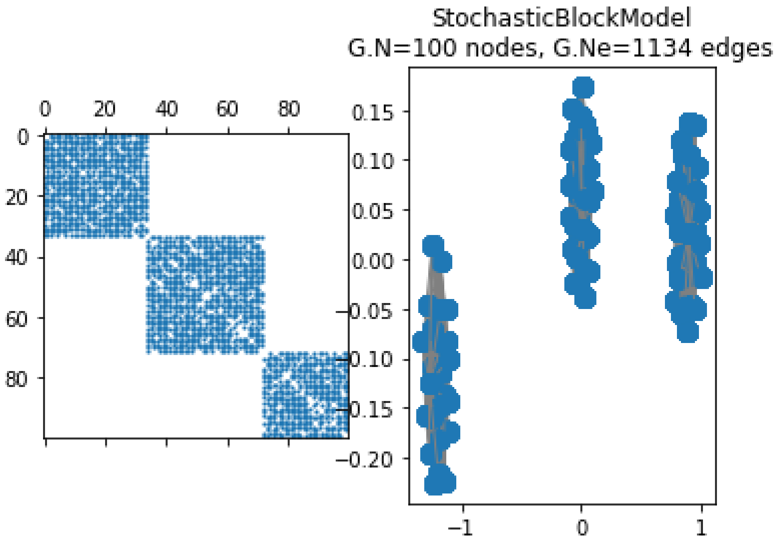
```
#importing all important libraries
from pygsp.graphs import Graph
from pygsp import utils
from pygsp import graphs
import numpy as np
from scipy import sparse
```

```
#implementing our custom stochastic block model
class StochasticBlockModel(Graph):
    #lets initialize our parameters
    def __init__(self, N=None,  self_loops=False, M=None, p=None, q=None, max_iter=10,dire
                 seed=None,k=None, z=None, **kwargs):

        #defining the parameters

        # k is the number of classes to be used
        # N is the given number of nodes
        # M  is the matrix containing the nodes probability
        # p is the diagonal values
        # q is the off diagonal values



        # setting the random binary edges for the triangle of the matrix
        binary_edges_t = np.random.RandomState(seed)

        edges=z
        if edges is None:
            edges = binary_edges_t.randint(0, k, N)
            edges.sort()

        #next is that we gonna generate stochastic block model matrix
        if M is None:

            p = np.asarray(p)
            if p.size == 1:
                num_classes=k
                p = p * np.ones(num_classes)
```

```
            if p.shape != (num_classes,):
                raise ValueError('Given parameter p is neither a scalar nor a vector.') #t
                #the matrix

            if q is None:
                q = 0.3 / num_classes
            q = np.asarray(q)
            if q.size == 1:
                q = q * np.ones((num_classes, num_classes))
            if q.shape != (num_classes, num_classes):
                raise ValueError('Given parameter q is neither a scalar nor a vector .')

            # re-setting the matrix containing the nodes probability equal to off diagonal
            M = q

            #function to edit the diagonal entries
            M.flat[::num_classes+1] = p

        if (M < 0).any() or (M > 1).any():
            raise ValueError('Values should be in range of [0, 1].')

        for iteration_val in range(max_iter):
            # getting the eigenvalues and eigenvectors of the matrix
            total_rows_val, total_columns_val = 0, 0
            data_val, i, csr_j = [], [], []
            for egn in range(N**2):
                if total_rows_val != total_columns_val or self_loops:
                    if total_rows_val >= total_columns_val or directed:
                        if binary_edges_t.uniform() < M[z[total_rows_val], z[total_columns
                            data_val.append(1)
                            i.append(total_rows_val)
                            j.append(total_columns_val)
                if total_rows_val < N-1:
                    total_rows_val += 1
                else:
                    total_rows_val = 0
                    total_columns_val += 1

            W = sparse.csr_matrix((data_val, (i,j)), shape=(N, N))

            #this is gonna be making the matrix symmetric
            if not directed:
                W = utils.symmetrize(W, method='tril')

            if not connected:
                break
            else:
                self.W = W
                self.A = (W > 0)
                if self.is_connected(recompute=True):
                    break
            if iteration_val == max_iter - 1:
                raise ValueError('Sorry, graph could not be fully connected after {} trial

        self.info = { z,np.bincount(z), np.sqrt(N)}
```

```
        #calling the SBM model
        model = 'StochasticBlockModel'
        super(StochasticBlockModel, self).__init__(gtype=model, W=W, **kwargs)
```

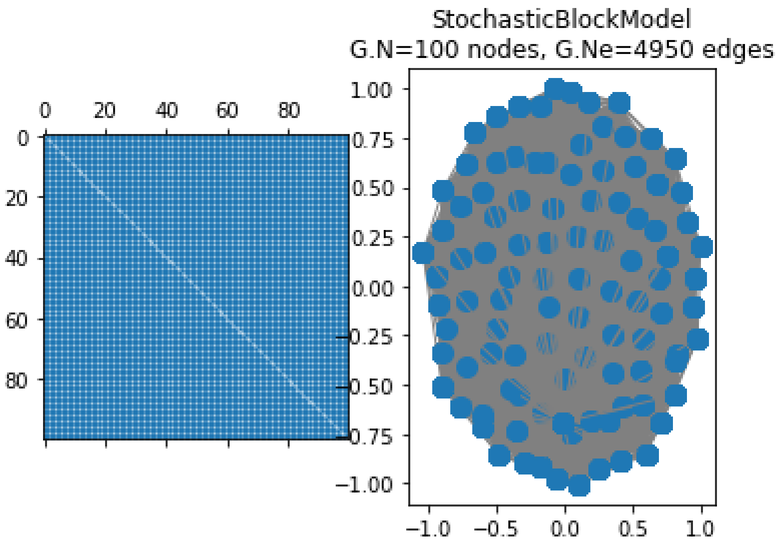
```
#we gonna be showing the  matrix, Fiedler vector and eigenvalues

# (i) plot for p=0.7 and  q=0

import matplotlib.pyplot as plt
Model = graphs.StochasticBlockModel(N=100,  p=0.7, q=0, seed=1,k=3)
Model.set_coordinates(seed=1)
fig, axes = plt.subplots(1,2)
_ = axes[0].spy(Model.W, markersize=1)
Model.plot(ax=axes[1])
```



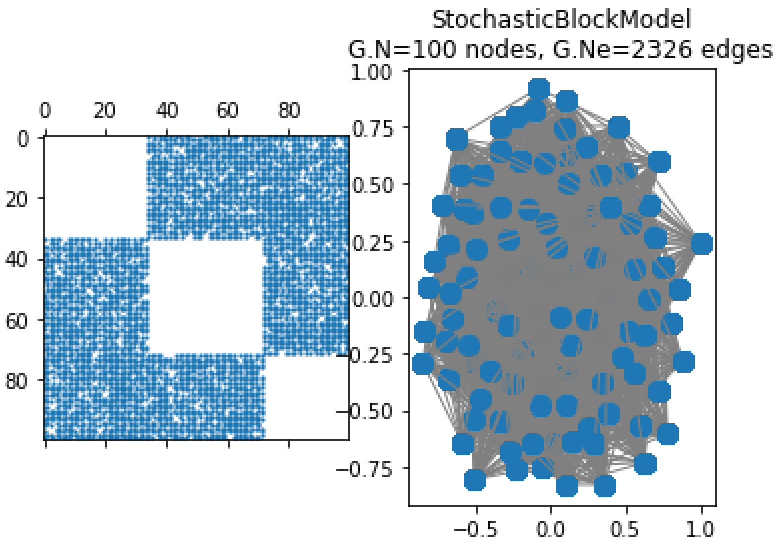

```
# (ii) plot for p=0.7 and  q=0.6

Model = graphs.StochasticBlockModel(N=100,  p=0.7, q=0.6, seed=1,k=3)
Model.set_coordinates(seed=1)
fig, axes = plt.subplots(1, 2)
_ = axes[0].spy(Model.W, markersize=1)
Model.plot(ax=axes[1])
```

StochasticBlockModel
G.N=100 nodes, G.Ne=3135 edges

```
# (iii) plot for p=1 and  q=1

Model = graphs.StochasticBlockModel(N=100,  p=1, q=1, seed=1,k=3)
Model.set_coordinates(seed=1)
fig, axes = plt.subplots(1, 2)
_ = axes[0].spy(Model.W, markersize=1)
Model.plot(ax=axes[1])
```




```
# (iv) plot for p=0 and  q=0.7

Model = graphs.StochasticBlockModel(N=100,  p=0, q=0.7, seed=1,k=3)
Model.set_coordinates( seed=1)
fig, axes = plt.subplots(1, 2)
_ = axes[0].spy(Model.W, markersize=1)
Model.plot(ax=axes[1])
```




```
#the kind of graph shown above is an inferring modular network graph with well deeply conr
```

END OF STOCHASTIC BLOCK MODEL IMPLEMENTATION AND TESTING. THANK YOU!!!

✓ 0s completed at 1:06 AM