

```
In [ ]: import networkx as nx
        from scipy.io import loadmat
        from scipy.linalg import eigh
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

a) Construct a network from the flight data

```
In [ ]: # Load data
        node_metadata = pd.read_csv('data/epidemics/airport_Nodes_GC.csv')
        edge_data = pd.read_csv('data/epidemics/airport_Edges_GC.csv')

        # map given node IDs to index order
        node_id_map = {given_id: node_id for (given_id, node_id) in zip(node_metadata.Id, node_
```

```
In [ ]: # build directed adjacency matrix from edge data
        A_dir = np.zeros((len(node_id_map), len(node_id_map)))

        for i, row in edge_data.iterrows():
            u, v = node_id_map[row.Source], node_id_map[row.Target]
            A_dir[u][v] = row.Weight

        # Compute undirected A
        A = (A_dir + A_dir.T) / 2
        G = nx.from_numpy_matrix(A)
```

```
In [ ]: # confirm graph is connected
        print('Graph is connected: {}'.format(nx.is_connected(G)))
```

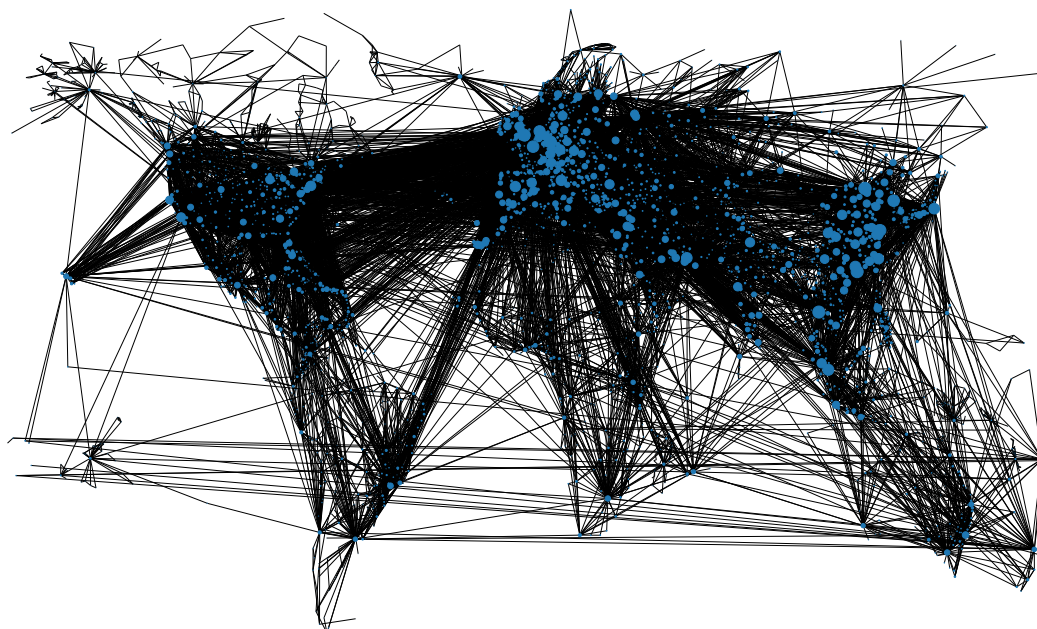
Graph is connected: True

b) Plot the airport network

```
In [ ]: # Compute top 2 e-val centralities as sanity check
        evec_centralities = nx centrality.eigenvector centrality(G, weight='weight')
        'Top 2 e-vec centralities: {}'.format(list(sorted(evec_centralities.values()))[-2:])
```

```
Out[ ]: 'Top 2 e-vec centralities: [0.16991022532548966, 0.1785292296913075]'
```

```
In [ ]: # plot the network, using lat/long as node locations and e-vector centrality as node si
        pos = {idx: (row.Longitude, row.Latitude) for idx, row in node_metadata.iterrows()}
        node_size = [evec_centralities[n] * 1750 for n in G.nodes()]
        plt.figure(1, figsize=(25, 15))
        nx.draw(G, pos=pos, node_size=node_size, width=0.25)
```



d) Mean-field approximation

In []:

```
def dp_dt(A, p_t, beta, gamma):
    """
    The provided formula for dp/dt, computed using matrix operations
    """
    left = beta * (1 - p_t)
    right = (A @ p_t) - (gamma * p_t)
    return left * right

def infect(A, beta, gamma, dp_dt):
    """
    Models the spread of infection throughout a graph over time

    beta, gamma: model parameters
    dp_dt: function to compute change in infection probabilities across nodes

    Returns P; P[a][b] = infection probability for ath node at bth timestep
    """
    # infect the first 20 nodes
    p_0 = np.zeros((len(A),))
    p_0[:20] = 1

    # compute infection probabilities over time
    t_max = 5
    delta = 0.05

    P = np.zeros((len(p_0), int(t_max / delta)))
    P[:, 0] = p_0

    for i in range(1, int(t_max / delta)):
        P[:, i] = P[:, i-1] + (delta * dp_dt(A, P[:, i-1], beta, gamma))

    return P
```

```

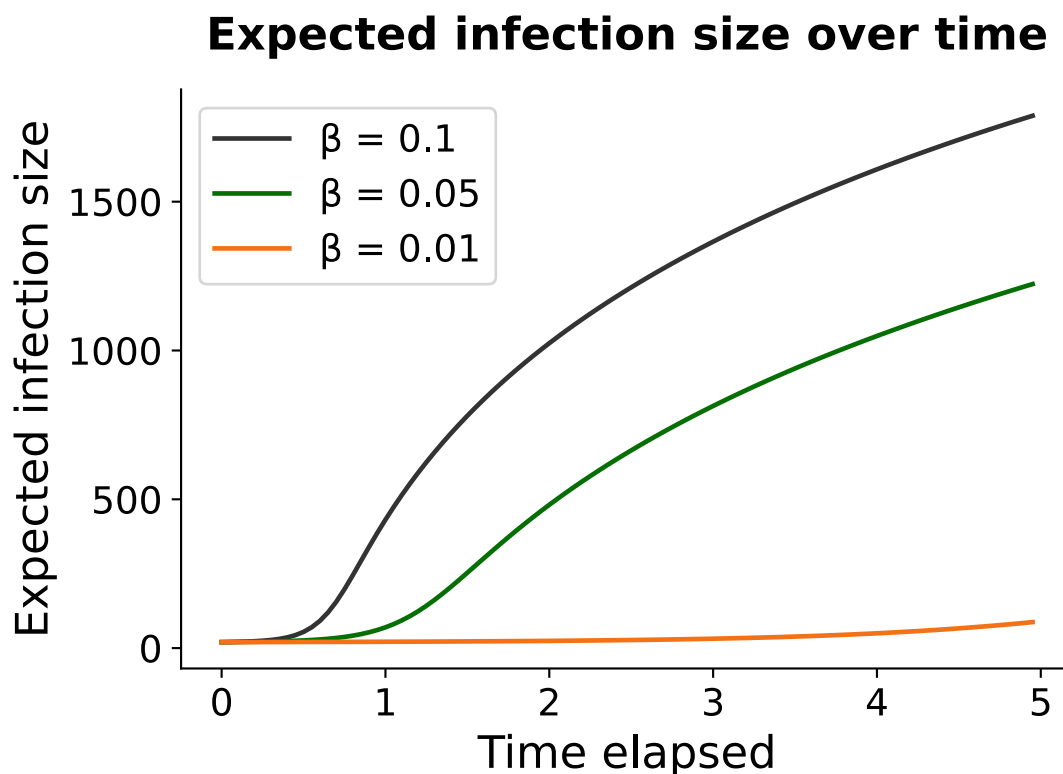
In [ ]: gamma = 0.1

# test varying values of beta
P_bar_all = []
for beta in (0.1, 0.05, 0.01):
    P = infect(A, beta, 0.1, dp_dt)
    P_bar_all.append(P.sum(axis=0))

# plot expected infected size over time for each beta value
x = np.array(range(0, int(5 / 0.05))) * 0.05
[plt.plot(x, P_bar) for P_bar in P_bar_all]
plt.legend([" $\beta = 0.1$ ", " $\beta = 0.05$ ", " $\beta = 0.01$ "])
plt.title("Expected infection size over time")
plt.xlabel("Time elapsed")
plt.ylabel("Expected infection size")

```

Out[]: Text(0, 0.5, 'Expected infection size')



e) Minimum number of immunizations

```

In [ ]: def immunize(A, node_order, eval_thresh, start_at=0):
    """
    Iteratively remove nodes, in the given order, until the largest e-value of the graph
    drops below the given threshold

    A: adjacency matrix
    node_order: order in which to remove nodes
    eval_thresh: terminate when largest e-val drops below this threshold
    start_at: remove this many nodes before starting to compute largest e-val (saves time)
    """

```

```

A_imm = A.copy()

for i in range(len(node_order)):
    # remove a node
    n = node_order[i]

    # setting to 0 creates additional e-vals that are 0, but still produces valid l
    # this is easier than deleting rows/columns
    A_imm[n, :] = 0
    A_imm[:, n] = 0

    if i < start_at:
        continue
    elif i == start_at:
        print('Starting with {} nodes removed'.format(i))

    # recompute max e-val
    w_max = eigh(A_imm, eigvals_only=True, eigvals=(len(A_imm) - 1, len(A_imm) - 1))

    if w_max < eval_thresh:
        print('Max e-val after removing {} nodes: {}'.format(i+1, w_max))
        print('Max e-val below threshold after removing {} nodes'.format(i+1))
        break

    if i % 5 == 4:
        print('Max e-val after removing {} nodes: {}'.format(i+1, w_max))

return A_imm

```

```

In [ ]: # compute required threshold for  $\lambda_{\max}(A)$ 
beta = 0.01
gamma = 0.4

print('Must have  $\lambda_{\max}(A) < \{\}$ '.format(gamma / beta))

```

Must have $\lambda_{\max}(A) < 40.0$

```

In [ ]: # compute degree centrality
degree_centralities = A.sum(axis=1) / (len(A) - 1)

# order nodes by degree centrality
node_order = np.argsort(degree_centralities)[::-1]

# compute number of nodes necessary to remove
immunize(A, node_order, gamma / beta, start_at=50); # start with 50 to speed up re-comp

```

Starting with 50 nodes removed
 Max e-val after removing 55 nodes: 45.751362453969314
 Max e-val after removing 60 nodes: 45.749493453262154
 Max e-val after removing 65 nodes: 45.74924054241361
 Max e-val after removing 70 nodes: 41.151565542207784
 Max e-val after removing 73 nodes: 37.41654106581537
 Max e-val below threshold after removing 73 nodes

```

In [ ]: # compute e-val centrality
eval_centralities_dict = nx centrality.eigenvector centrality(G, weight='weight')
eval_centralities = np.array([eval_centralities_dict[n] for n in G.nodes()])

```

```
# order nodes by e-val centrality
node_order = np.argsort(eval_centralities)[::-1]

# compute number of nodes necessary to remove
immunize(A, node_order, gamma / beta, start_at=100);
```

Starting with 100 nodes removed

Max e-val after removing 105 nodes: 48.51744278279449

Max e-val after removing 110 nodes: 48.51591111485003

Max e-val after removing 115 nodes: 48.51590896694197

Max e-val after removing 120 nodes: 48.51590773253301

Max e-val after removing 125 nodes: 39.427119599712015

Max e-val below threshold after removing 125 nodes

Degree centrality only required 73 nodes to be removed, while e-val centrality required 125 nodes to be removed.