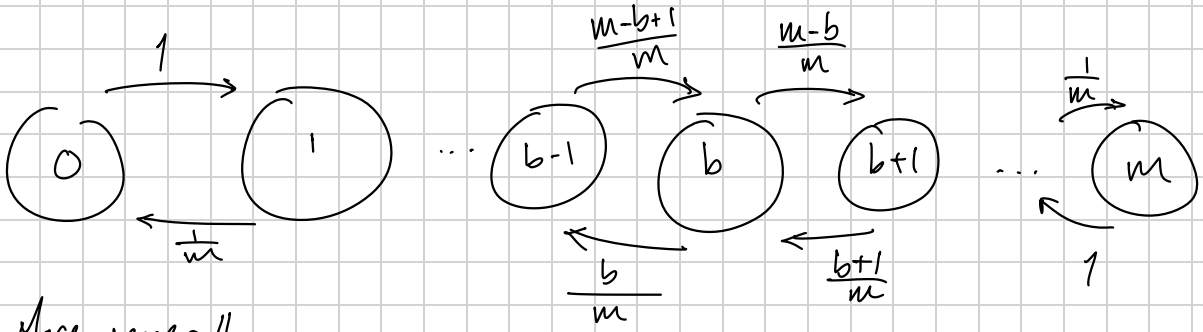
1)

Spot: b fleas

Lassie: $m-b$ fleas

States: possible number of fleas on Spot $= \{0, 1, \dots m\}$



More generally,

$$P \in \mathbb{R}^{(m+1)\times(m+1)} : P(i,j) = \begin{cases} \frac{m-i}{m}, & j = i+1 \\ \frac{i}{m}, & j = i-1 \\ 0 & \text{else} \end{cases}$$

(0-indexed)

$$= \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ \frac{1}{m} & 0 & \frac{m-1}{m} & & & \\ 0 & \frac{2}{m} & 0 & \ddots & & \\ & & & & 0 & \frac{1}{m} \\ & & & & 1 & 0 \end{bmatrix}$$

$$\pi_k P^{(i)}(k, k+1) = \pi_{k+1} P^{(j)}(k+1, k) \quad \forall k = 0, \dots, m-1$$

$$\pi_k \frac{m-k}{m} = \pi_{k+1} \frac{k+1}{m}$$

$$\pi_{k+1} = \pi_k \frac{m-k}{k+1} \quad \forall k \dots$$

must be normalized

$$\Rightarrow \pi_k = \left(\pi_0 \frac{m}{1}\right)\left(\frac{m-1}{2}\right) \cdots \left(\frac{m-k+1}{k}\right)$$

$$\Rightarrow c\pi_k = \pi_0 \frac{m!}{k!\,(m-k)!}$$

$$\sum_k \pi_k = \sum_{k=0}^{m} \pi_0 \binom{m}{k} = \pi_0 2^m \triangleq 1$$

$$\Rightarrow \pi_0 = \frac{1}{2^m} \Rightarrow \pi_k = \frac{\binom{m}{k}}{2^m}$$

$\sim Bin(n = m, p = 1/2)$

2)

Let $G = (V, E)$ be a graph with MC X_0 (random walk)

a. Claim: X_0 irreducible $\iff$ G is connected

$\Rightarrow$) Let X_0 be irreducible

$\Rightarrow \exists\, n > 0 : P^n(i,j) > 0 \quad \forall i, j \in V$

Assume OTC that G is disconnected

$\Rightarrow \exists\, i, j \in V$: i and j are in two different connected components

$\Rightarrow \nexists$ path from i to j

$\Rightarrow P^n(i,j) = 0 \quad \forall n > 0$ ($\Rightarrow\Leftarrow$ with irreducibility)

$\therefore$ X_0 irreducible $\Rightarrow$ G connected ■

$\Leftarrow$) Let G be connected

By def of connected,
$\exists\, (i = v_0, v_1, \ldots, v_n = j) \quad \forall i, j \in V : (v_k, v_{k+1}) \in E$ for $k = 0, \ldots, n-1$

$\Rightarrow \exists$ a finite-length path between any two nodes

$\Rightarrow \exists\, n > 0 : P^n(i,j) > 0 \quad \forall i, j \in V$

$\therefore$ G connected $\Rightarrow$ X_0 irreducible ■

b. Let X_0 be irreducible and $G = (V, E)$ where $|V| \geq 2$ ←

Claim: X_0 aperiodic $\iff$ G is non-bipartite

$\Rightarrow$) Let X_0 be aperiodic

(I) $\Rightarrow \gcd\{n : P^n(i,i) > 0\} = 1 \quad \forall i \in V$

Consider arbitrary $i \in V$

Since X_0 is irreducible, $(i,j) \in E$ for some $j \in V$.

(II) $\Rightarrow$ $\exists$ a 2-length cycle $((i,j), (j,i))$ from i to i.

(I, II) $\Rightarrow$ $\exists$ odd-length cycle from i to i

$\Rightarrow$ G is non-bipartite ∎

$\Leftarrow$) Let G be non-bipartite

$\Rightarrow$ $\exists$ odd-length $2k+1$ cycle at some $i \in V$, $k \in \mathbb{Z} \geq 0$

Since X_0 is irreducible, $(i,j) \in E$ for some $j \in V$.

$\Rightarrow$ $\exists$ a 2-length cycle $((i,j), (j,i))$ from i to i.

$\gcd\{2, 2k+1\} \geq \gcd(\{2, 2k+1\} \cup S) \geq 1$ for any set S

$\Rightarrow \gcd\{2, 2k+1\} \geq \gcd\{n : P^n(i,i) > 0\} \geq 1$

$\Rightarrow \quad 1 \geq \gcd\{n : P^n(i,i) > 0\} \geq 1$

$\Rightarrow \gcd\{n : P^n(i,i) > 0\} = 1$

$\Rightarrow$ X_0 is aperiodic ∎

c. Claim: $\Pi(x) = \frac{\deg(x)}{2|\mathcal{E}|}$ is a stationary distribution of X_0.

$$P(i,j) = \begin{cases} \frac{1}{\deg(i)}, & (i,j) \in \mathcal{E} \\ 0, & \text{else} \end{cases}$$

Want to prove $\Pi P = \Pi$ or equivalently,

$$\Pi(x) = \sum_i \Pi(i) P(i,x) \qquad \forall\ x = 1, 2, \dots, |V|$$

$$= \sum_i \frac{\deg i}{2|\mathcal{E}|} P(i,x) = \sum_{\{i : (i,x) \in \mathcal{E}\}} \frac{\cancel{\deg i}}{2|\mathcal{E}|} \frac{1}{\cancel{\deg i}}$$

$$= \frac{\deg x}{2|\mathcal{E}|} = \Pi(x) \qquad \blacksquare$$

In addition, $\sum_{i \in V} \Pi(i) = \sum_{i \in V} \frac{\deg(i)}{2|\mathcal{E}|} = \frac{2|\mathcal{E}|}{2|\mathcal{E}|} = 1$

$\Rightarrow \Pi$ is a valid distribution $\blacksquare$

Markov Chain

March 4, 2025

[15]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
```

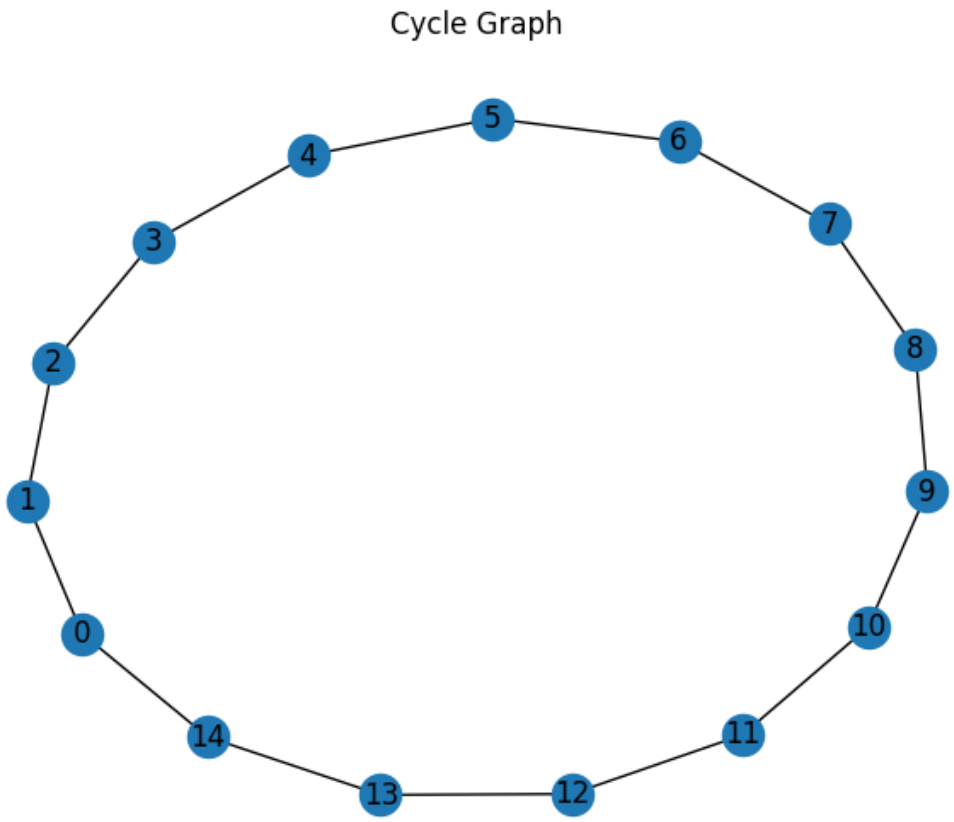
In this notebook, you'll determine the relation between mixing time and the number of nodes in a graph.

0.1 networkx

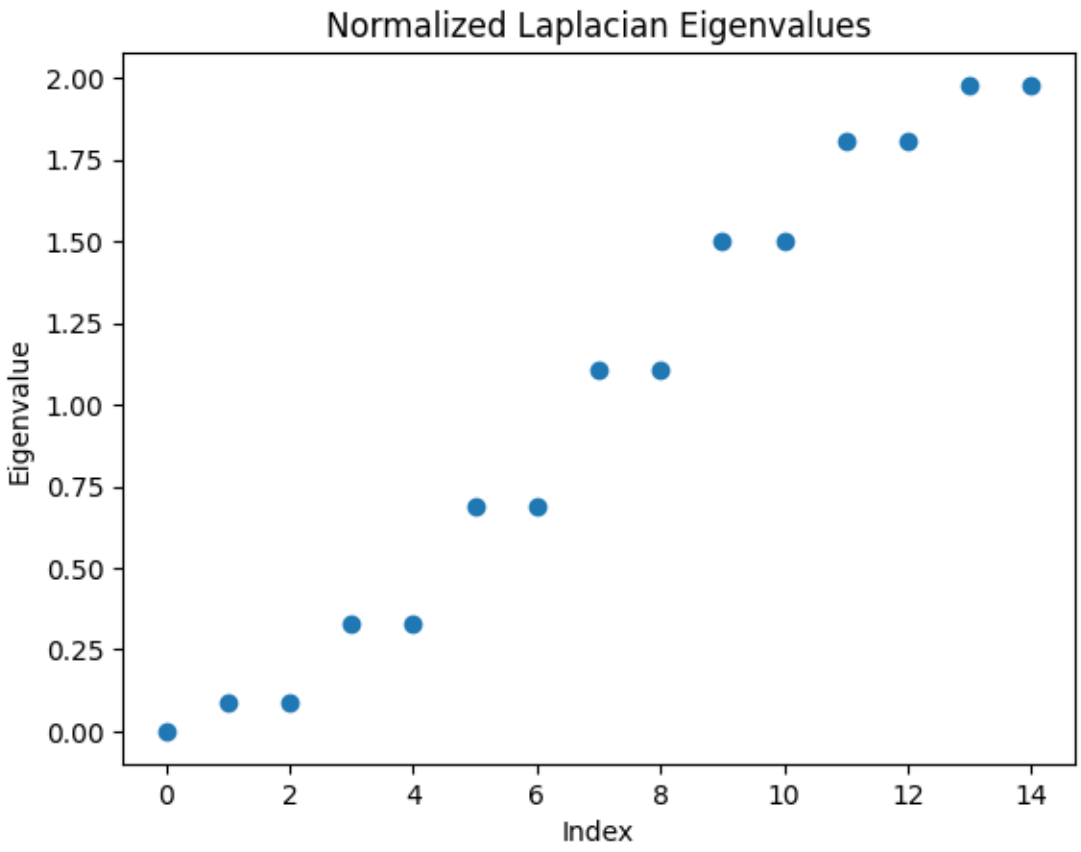
The first task to get some familiarity with `networkx`. For all of the tasks below, use the `networkx` library.

1. Create a cycle graph with 15 vertices.
2. Draw the graph.
3. Print its number of nodes, number of edges, the adjacency matrix.
4. Plot the eigenvalues of the normalized Laplacian matrix in increasing order. You'll need to use `todense()` to convert the sparse matrix produced by `networkx.normalized_laplacian_matrix()` to a dense matrix.

[16]:
```
# Test the networkx library
G: nx.Graph = nx.cycle_graph(15)
nx.draw(G, with_labels=True)
plt.title('Cycle Graph')
plt.show()
print("Number of nodes =", len(G.nodes))
print("Number of edges =", len(G.edges))
print("Adjacency matrix = \n", nx.adjacency_matrix(G).todense())
lap_adj_matrix = nx.normalized_laplacian_matrix(G).todense()
eigvals = sorted(np.linalg.eigvals(lap_adj_matrix))
plt.plot(eigvals, 'o')
plt.title('Normalized Laplacian Eigenvalues')
plt.xlabel('Index')
plt.ylabel('Eigenvalue')
plt.show()
```




```
Number of nodes = 15
Number of edges = 15
Adjacency matrix =
 [[0 1 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [1 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 1]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 1 0]]
```



0.2 GraphMC

Define the following methods for the class `GraphMC`. Instead of finding the `mixing_time` over all possible distributions, we'll simply find the mixing time from a given initial distribution.

[17]:
```
class GraphMC(nx.Graph):
    def __init__(self, graph):
        super().__init__(graph)  # Initialize from an existing graph
        self.trans_matrix = nx.adjacency_matrix(self).todense() / np.sum(nx.
 ↪adjacency_matrix(self), axis=1)

    def transition_probabilities(self, start_node, end_node):
        assert start_node in self.nodes, "Start node must be in the graph"
        assert end_node in self.nodes, "End node must be in the graph"
        return self.trans_matrix[start_node, end_node]

    def stationary_distribution(self):
        eigvals, eigvecs = np.linalg.eig(self.trans_matrix.T)
        idx = np.argmin(np.abs(eigvals - 1.0))
        stat_dist = np.abs(eigvecs[:, idx])
```

```
        return stat_dist / np.sum(stat_dist)

    def update(self, distribution, num_steps=1):
        assert len(distribution) == len(self.nodes), "Distribution length must␣
↪match number of nodes"
        return distribution @ np.linalg.matrix_power(self.trans_matrix,␣
↪num_steps)

    def total_variational_distance(self, distribution):
        assert len(distribution) == len(self.nodes), "Distribution length must␣
↪match number of nodes"
        stationary_dist = self.stationary_distribution()
        return 0.5 * np.sum(np.abs(stationary_dist - distribution))

    def mixing_time(self, starting_distribution, epsilon=0.25, max_steps=1000):
        assert len(starting_distribution) == len(self.nodes), "Distribution␣
↪length must match number of nodes"
        steps = 0
        x = starting_distribution.copy()
        while self.total_variational_distance(x) > epsilon and steps <=␣
↪max_steps:
            x = self.update(x)
            steps += 1
        return steps
```

0.3 Test 1

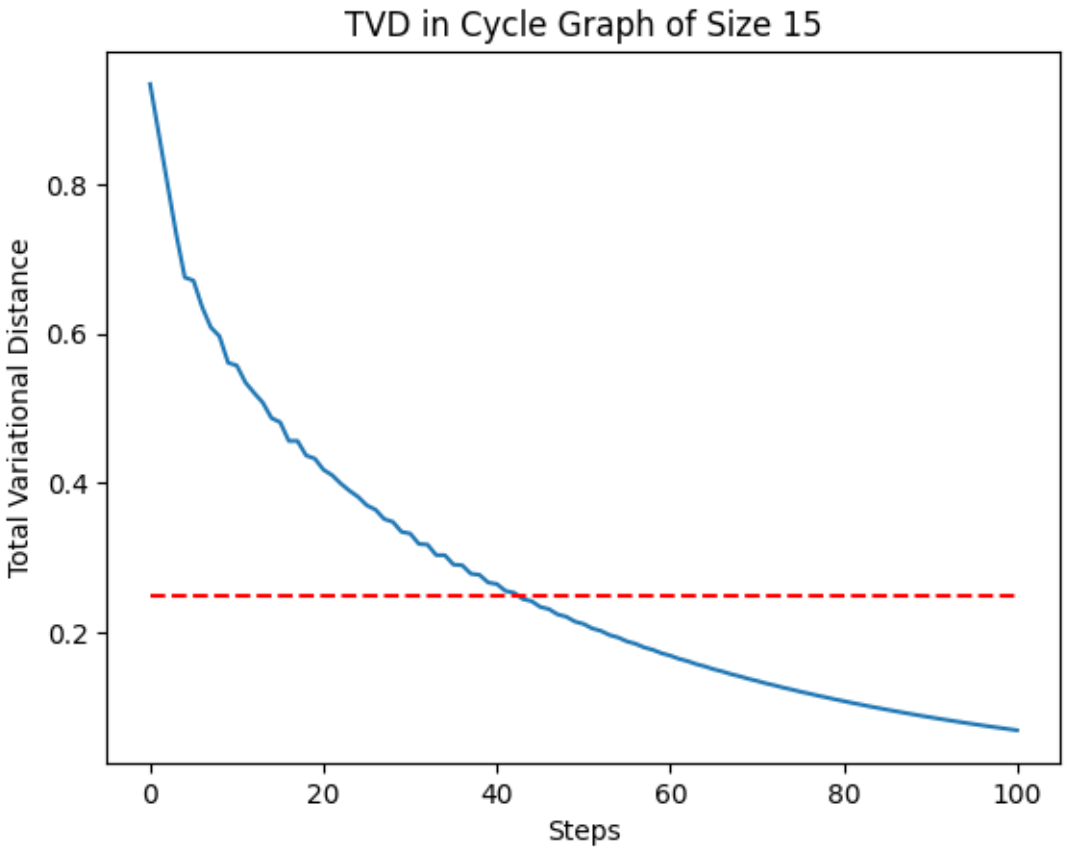
Test your class on a cycle graph with 15 vertices. Start with the distribution where all the probability is concentrated on a single node.

1. Run the markov chain for 100 steps and compute the total variation distance between the distribution and the stationary distribution.
2. Plot the total variation distance as a function of the number of steps.
3. Mark the threshold $\epsilon = 0.25$ on the plot.
4. Print the mixing time.

[18]:
```
# Test 1
G: nx.Graph = nx.cycle_graph(15)
mc = GraphMC(G)
eps = 0.25
num_steps = 100

x = np.zeros(len(G.nodes))
x[0] = 1
dist = [None] * (num_steps + 1)
dist[0] = x
```

```python
for i in range(0, num_steps):
    dist[i+1] = mc.update(dist[i], 1)
var_dist = [mc.total_variational_distance(d) for d in dist]

print(f"Final TVD = {var_dist[-1]:.4f}")
print("Mixing time =", mc.mixing_time(x, epsilon=eps))

plt.plot(var_dist)
plt.title('TVD in Cycle Graph of Size 15')
plt.xlabel('Steps')
plt.ylabel('Total Variational Distance')
plt.plot([eps] * len(var_dist), 'r--')
plt.show()
```

```
Final TVD = 0.0700
Mixing time = 43
```



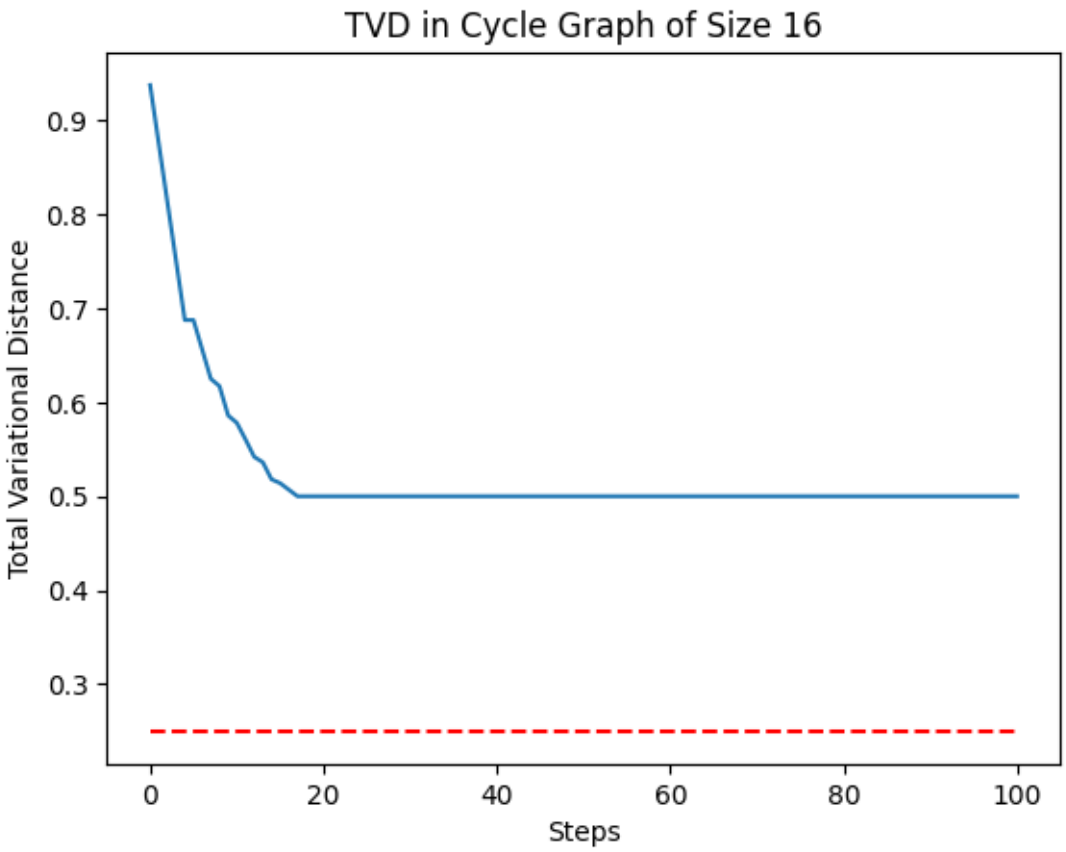
0.4 Test 2

Test your class on a cycle graph with 16 vertices. Start with the distribution where all the probability is concentrated on a single node.

1. Run the markov chain for 100 steps and compute the total variation distance between the distribution and the stationary distribution.
2. Plot the total variation distance as a function of the number of steps.
3. Mark the threshold $\epsilon = 0.25$ on the plot.

What do you observe? Why is this case different from the previous one?

[19]:
```
# Test 2
G: nx.Graph = nx.cycle_graph(16)
mc = GraphMC(G)
eps = 0.25
num_steps = 100

x = np.zeros(len(G.nodes))
x[0] = 1
dist = [None] * (num_steps + 1)
dist[0] = x

for i in range(0, num_steps):
    dist[i+1] = mc.update(dist[i], 1)
var_dist = [mc.total_variational_distance(d) for d in dist]

plt.plot(var_dist)
plt.title('TVD in Cycle Graph of Size 16')
plt.xlabel('Steps')
plt.ylabel('Total Variational Distance')
plt.plot([eps] * len(var_dist), 'r--')
plt.show()
```



While the previous graph has a TVD that keeps decreasing to approach 0, here the TVD levels off at 0.5. Even-sized cycle graphs like this one are bipartite, meaning that any random walk will always alternate between "red-colored" and "blue-colored" nodes. As the number of steps approaches infinity, the Markov chain's probability distribution approaches an oscillation between uniform across even-indexed nodes, and uniform across odd-indexed nodes. Since these distributions have an absolute fractional error of 1 at each node and the sum of a distribution is 1, the TVD approaches 0.5.

0.5 Mixing time as a function of cycle length and spectral gap

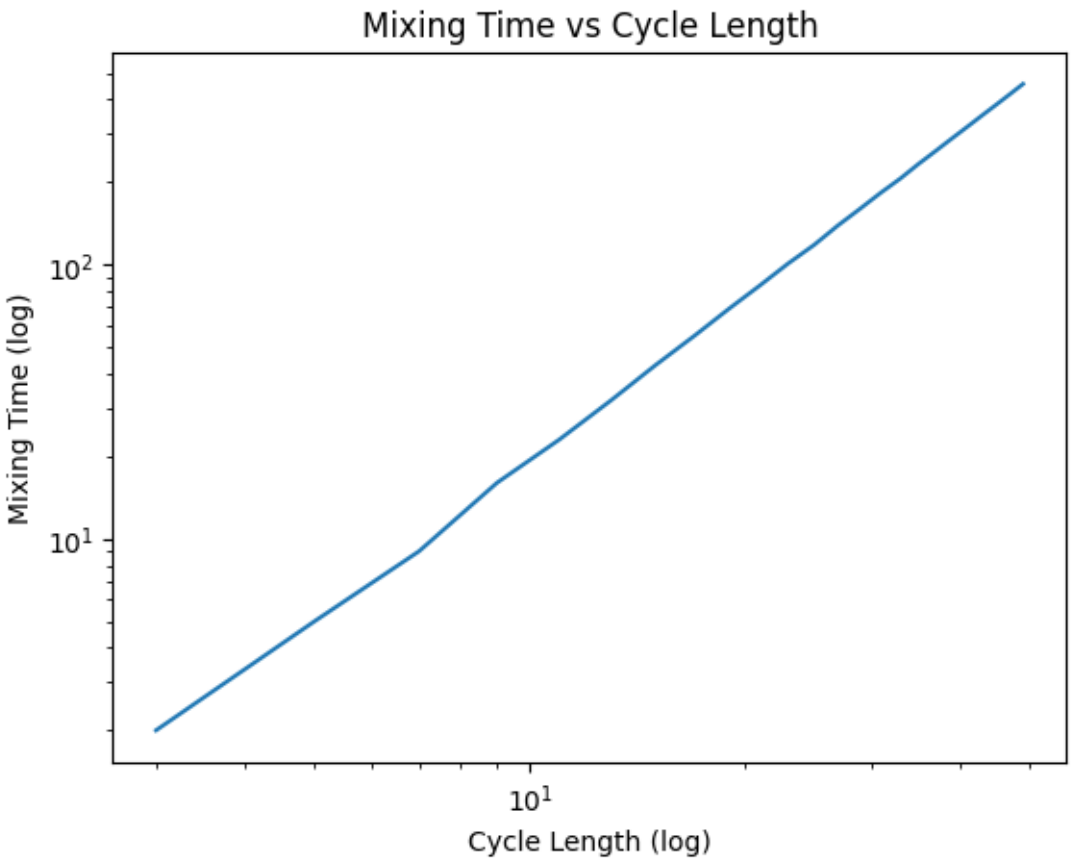
Now vary the length of the cycle graph over odd numbers from 3 to 51.

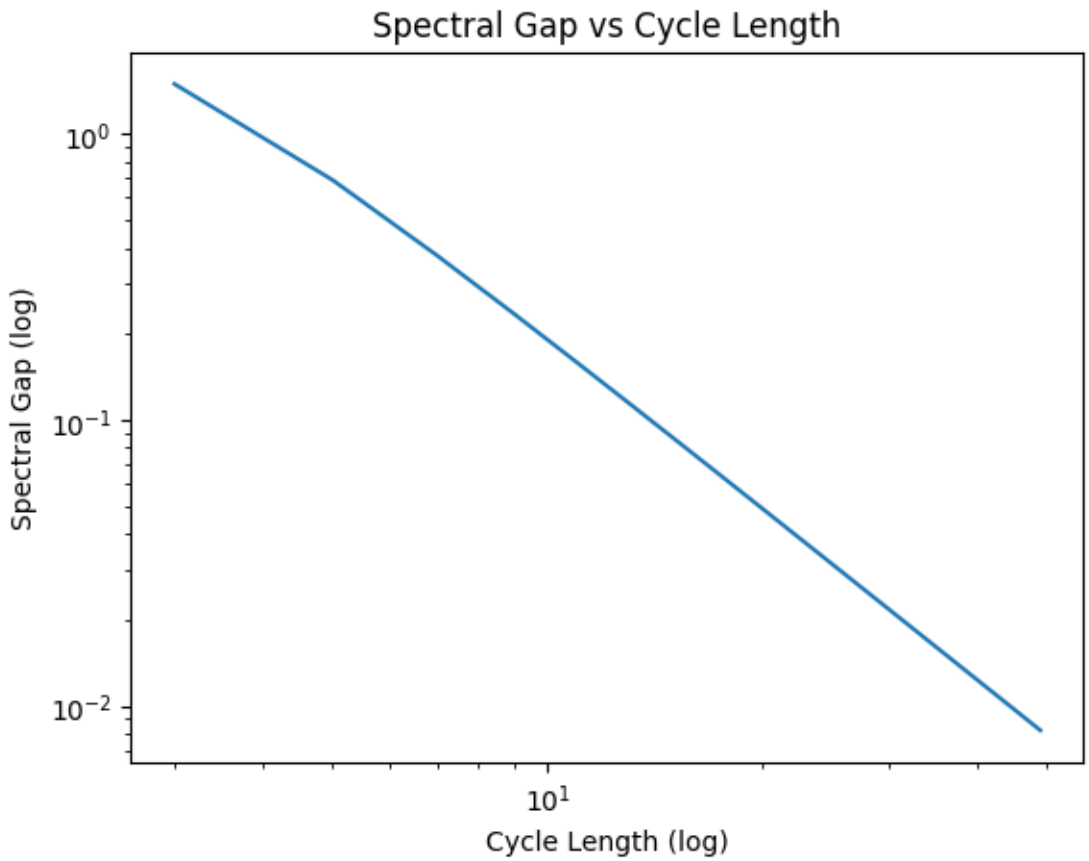
1. Compute the mixing time for each cycle graph with the initial distribution where all the probability is concentrated on a single node.
2. Compute the spectral gap for each cycle graph. The spectral gap is the second smallest eigenvalue of the normalized Laplacian matrix.
3. Plot the mixing time vs the cycle length on a log-log scale. Find the rate of growth of the mixing time as a function of the number of nodes i.e. find α in the relation $mixing\_time = O(n^{\alpha})$.
4. Plot the mixing time vs the spectral gap on a log-log scale. Find the rate of growth of the mixing time as a function of the spectral gap i.e. find β in the relation $mixing\_time =$

$O(gap^{\beta})$.

[20]:
```
# Mixing time as a function of cycle length and spectral gap
num_nodes = np.arange(3, 51, 2)
mixing_time = np.zeros(len(num_nodes))
spectral_gap = np.zeros(len(num_nodes))

for i, n in enumerate(num_nodes):
    G: nx.Graph = nx.cycle_graph(n)
    mc = GraphMC(G)
    x = np.zeros(len(G.nodes))
    x[0] = 1
    mixing_time[i] = mc.mixing_time(x, epsilon=0.25)
    norm_lap = nx.normalized_laplacian_matrix(G).todense()
    eigvals = sorted(np.abs(np.linalg.eigvals(norm_lap)))
    spectral_gap[i] = eigvals[1]

reg_mixing_time = np.polyfit(np.log(num_nodes), np.log(mixing_time), 1)
reg_spectral_gap = np.polyfit(np.log(num_nodes), np.log(spectral_gap), 1)

a, b = reg_mixing_time[0], reg_spectral_gap[0]
print(f"Mixing time order: O(n^{a:.2f}), a = {a:.2f}")
print(f"Spectral gap order: O(n^{b:.2f}), b = {b:.2f}")

plt.plot(num_nodes, mixing_time)
plt.xscale('log')
plt.yscale('log')
plt.title('Mixing Time vs Cycle Length')
plt.xlabel('Cycle Length (log)')
plt.ylabel('Mixing Time (log)')
plt.show()

plt.plot(num_nodes, spectral_gap)
plt.xscale('log')
plt.yscale('log')
plt.title('Spectral Gap vs Cycle Length')
plt.xlabel('Cycle Length (log)')
plt.ylabel('Spectral Gap (log)')
plt.show()
```

```
Mixing time order: O(n^1.97), a = 1.97
Spectral gap order: O(n^-1.92), b = -1.92
```


Mixing Time vs Cycle Length
Mixing Time (log)
10^2
10^1
10^1
Cycle Length (log)


Spectral Gap vs Cycle Length
Spectral Gap (log)
10^{0}
10^{-1}
10^{-2}
10^{1}
Cycle Length (log)