StrategicInfluenceCycles

July 11, 2021

1 3 Agent Cycle Case

James Yu

11 July 2021

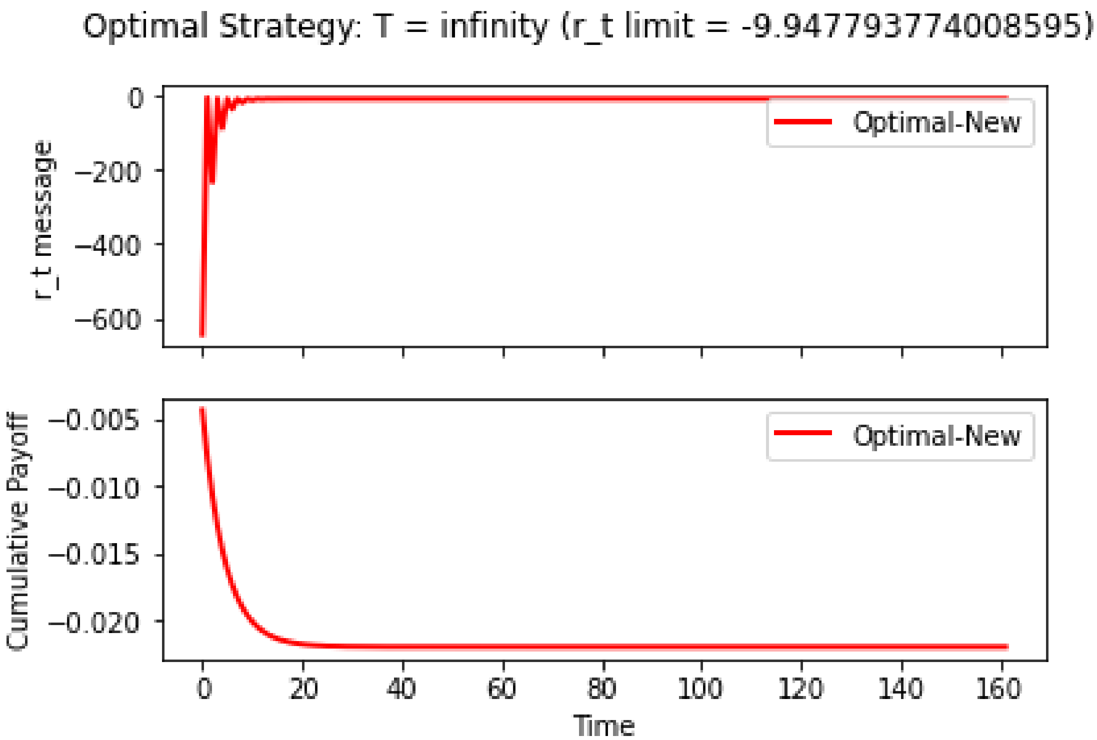
[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
def optimal_K(z, delta = 0.8):
    # the network:
    A = np.array([
      [0, 0.5, 0.5],
      [0.99, 0, 0],
      [0.99, 0, 0],
    ], ndmin = 2)

    # the bot fractions
    c = np.array([0, 0.01, 0], ndmin = 2).T
    A_tilde = np.concatenate((np.concatenate((A, c), axis = 1), # A c
        np.concatenate((np.zeros((1, 3)), np.array([1], ndmin = 2)), axis =␣
↪1)), # 0 1
        axis = 0)

    # the strategic agent fractions
    B = np.array([0, 0, 0.01], ndmin = 2).T
    B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)

    # the initial opinions
    x = np.array([5, 0, 0], ndmin = 2).T
    w_0 = np.concatenate((x, np.array([z], ndmin = 2)), axis = 0)

    Q = 0.2 * np.identity(3)
    Q_tilde = 0.2 * np.identity(4)
    Q_tilde[3, :] = 0

    def L(K_entry):
        return -1 * np.linalg.inv(B_tilde.T @ K_entry @ B_tilde) @ B_tilde.T @␣
↪K_entry @ A_tilde
```

```python
    # first compute the sequence of optimal K_t matrices
    K = np.zeros((4, 4))
    K_t = [Q_tilde, K]
    K = Q_tilde
    current_difference = np.inf
    while abs(current_difference) != 0:
        K_new = delta * (A_tilde.T @ (K
                - (K @ B_tilde @ np.linalg.inv(B_tilde.T @ K @ B_tilde) @␣
↪B_tilde.T @ K))
                @ A_tilde) + Q_tilde
        K_t.insert(0, K_new)
        current_difference = np.max(np.abs(K - K_new))
        K = K_new

    # compute the Gamma matrix to use for later computations
    expr = A_tilde + B_tilde @ L(K_t[0])
    A_tilde_n = expr[:3, :3]
    c_nplus1 = np.array(expr[:3, 3], ndmin = 2).T
    x_t = x
    x_ts = [x]

    # compute the resulting sequence of x_t opinion vectors
    for K_ent in K_t:
        x_tp1 = A_tilde_n @ x_t + c_nplus1 * z
        x_ts.append(x_tp1)
        x_t = x_tp1

    # compute the sequence of r_t and cumulative costs
    payoff = 0
    payoffs = []
    r_ts = []
    i = 0
    for x_ent in x_ts:
        r_ts.append(L(K_t[0]) @ np.concatenate((x_ent, np.array([z], ndmin =␣
↪2)), axis = 0))
        payoff += (-1 * delta**i * (x_t.T @ Q @ x_t)).item() # account for␣
↪discounting
        payoffs.append(payoff)
        i += 1

    return r_ts, A_tilde, B_tilde, w_0, K_t, x_ts, payoffs
```

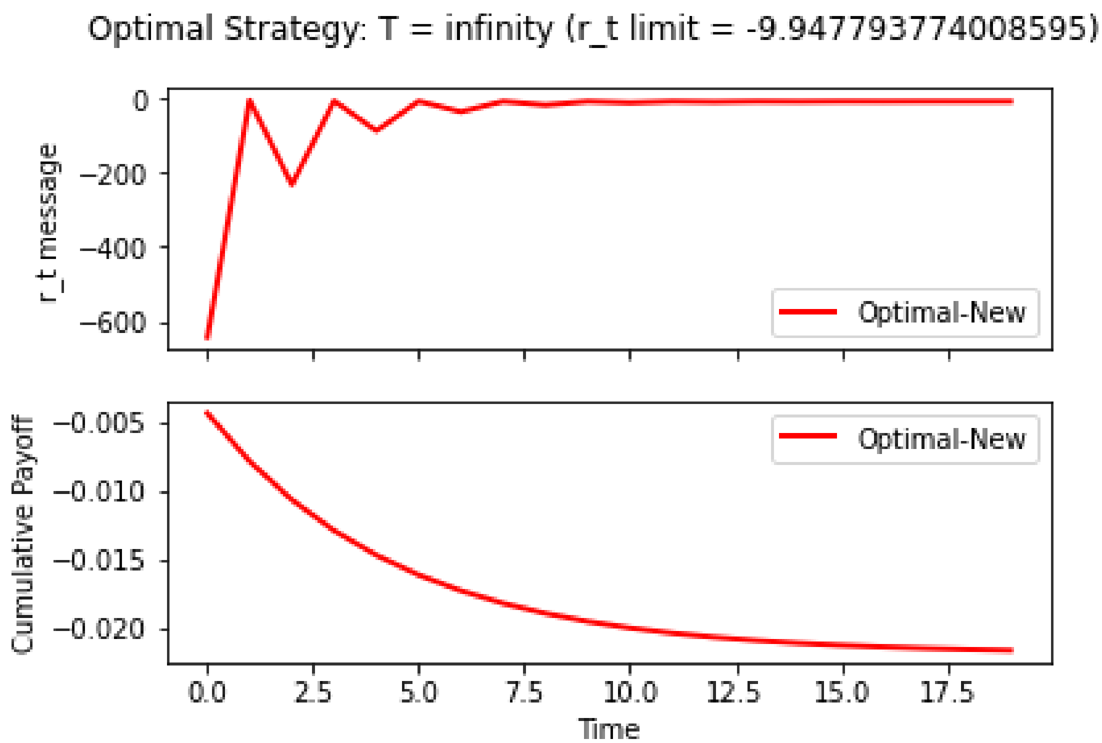
[3]:
```python
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K(10) # this means z = 10
fig, sub = plt.subplots(2, sharex=True)
fig.suptitle(f"Optimal Strategy: T = infinity (r_t limit = {rs[-1].item()})")
```

```
sub[0].plot(range(len(Ks)+1), [a.item() for a in rs], 'r', label =␣
 ↪"Optimal-New", linewidth=2)
sub[0].set(ylabel = "r_t message")
sub[1].plot(range(len(Ks)+1), ps, 'r', label = "Optimal-New", linewidth=2)
sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")

sub[0].legend()
sub[1].legend()
plt.show()
```



The first 20 time periods look interesting, so I can plot those separately:

[4]:
```
fig, sub = plt.subplots(2, sharex=True)
fig.suptitle(f"Optimal Strategy: T = infinity (r_t limit = {rs[-1].item()})")
sub[0].plot(range(20), [a.item() for a in rs[:20]], 'r', label = "Optimal-New",␣
 ↪linewidth=2)
sub[0].set(ylabel = "r_t message")
sub[1].plot(range(20), ps[:20], 'r', label = "Optimal-New", linewidth=2)
sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")

sub[0].legend()
sub[1].legend()
plt.show()
```



[5]:
```
print([a.item() for a in rs[:60]])
```

```
[-643.495755183268, -6.622921063423858, -233.6707898356107, -8.773690896807535,
-88.9504613154594, -9.533186152530849, -37.84578185989329, -9.801384559683846,
-19.799331240535412, -9.896092703095356, -13.42663927912538, -9.929536721364034,
-11.176268627765404, -9.941346712750896, -10.381601617470775,
-9.945517141538408, -10.100983108306979, -9.946989833236648,
-10.001889089472451, -9.947509880682263, -9.96689629944574, -9.947693523562856,
-9.954539394677369, -9.947758372852812, -9.950175835917461, -9.947781272898785,
-9.948634944779425, -9.947789359527118, -9.948090814258919, -9.947792215135262,
-9.947898666990344, -9.947793223528073, -9.947830814569961, -9.947793579618969,
-9.947806854037053, -9.947793705364337, -9.947798392921975, -9.947793749768444,
-9.947795405072386, -9.947793765448738, -9.94779434998153, -9.947793770985877,
-9.947793977400286, -9.947793772941193, -9.947793845831725, -9.947793773631668,
-9.947793799371295, -9.94779377387549, -9.947793782964855, -9.947793773961592,
-9.947793777171293, -9.947793773991998, -9.947793775125431, -9.947793774002733,
-9.947793774402982, -9.947793774006524, -9.947793774147863, -9.947793774007865,
-9.947793774057775, -9.947793774008337]
```

[6]:
```
def l_matrix(r_ss, A_tilde, B_tilde, w_0):
    A_tilde_prime = np.concatenate((np.concatenate((A_tilde, B_tilde), axis =␣
↪1), # A c
```

```
                       np.concatenate((np.zeros((1, 4)), np.array([1], ndmin =␣
↪2)), axis = 1)), # 0 1
                       axis = 0)
    w_0_prime = np.concatenate((w_0, np.array([r_ss], ndmin = 2)), axis = 0)
    print(A_tilde_prime)
    print()
    print(np.linalg.matrix_power(A_tilde_prime, 2))
    print(np.linalg.matrix_power(A_tilde_prime, 3))
    print(np.linalg.matrix_power(A_tilde_prime, 4))
    res_mat = np.linalg.matrix_power(A_tilde_prime, 1000000000000)
    res_vec = res_mat @ w_0_prime
    return res_mat, res_vec
```

The limit matrix, demonstrating that each naive agent is weighted toward whichever of the bot and strategic agent it supports by a factor of exactly 0.01/2:

[7]:
```
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.   0.5  0.5  0.   0.  ]
 [0.99 0.   0.   0.01 0.  ]
 [0.99 0.   0.   0.   0.01]
 [0.   0.   0.   1.   0.  ]
 [0.   0.   0.   0.   1.  ]]

[[0.99  0.    0.    0.005 0.005]
 [0.    0.495 0.495 0.01  0.   ]
 [0.    0.495 0.495 0.    0.01 ]
 [0.    0.    0.    1.    0.   ]
 [0.    0.    0.    0.    1.   ]]
[[0.      0.495   0.495   0.005   0.005  ]
 [0.9801  0.      0.      0.01495 0.00495]
 [0.9801  0.      0.      0.00495 0.01495]
 [0.      0.      0.      1.      0.     ]
 [0.      0.      0.      0.      1.     ]]
[[0.9801  0.      0.      0.00995 0.00995]
 [0.      0.49005 0.49005 0.01495 0.00495]
 [0.      0.49005 0.49005 0.00495 0.01495]
 [0.      0.      0.      1.      0.     ]
 [0.      0.      0.      0.      1.     ]]
[[0.    0.    0.    0.5   0.5  ]
 [0.    0.    0.    0.505 0.495]
 [0.    0.    0.    0.495 0.505]
 [0.    0.    0.    1.    0.   ]
 [0.    0.    0.    0.    1.   ]]
```

The first of these is the base matrix, the second/third/fourth are the second/third/fourth powers, and the last matrix is the actual limit matrix.

[8]:
```
print(xs[-1])
```

```
[[ 0.02610311]
 [ 0.12584208]
 [-0.07363586]]
```

The first agent has their agenda reduced drastically, the second is weighted toward the bot (as they listen to the bot) and the third is weighted, to a lesser extent, in the opposite direction.

[9]:
```
print("\n".join([str(a.flatten()) for a in xs]))
```

```
[5 0 0]
[ 0.          5.05       -1.48495755]
[ 1.78252122  0.1        -0.06622921]
[ 0.01688539  1.86469601 -0.57201189]
[ 0.64634206  0.11671654 -0.07102037]
[ 0.02284809  0.73987864 -0.24962597]
[ 0.24512634  0.12261961 -0.07271226]
[ 0.02495367  0.34267507 -0.13578275]
[ 0.10344616  0.12470414 -0.07330971]
[ 0.02569722  0.2024117  -0.09558161]
[ 0.05341504  0.12544024 -0.07352068]
[ 0.02595978  0.15288089 -0.0813855 ]
[ 0.0357477   0.12570018 -0.07359519]
[ 0.0260525   0.13539022 -0.07637247]
[ 0.02950888  0.12579197 -0.07362149]
[ 0.02608524  0.12921379 -0.07460223]
[ 0.02730578  0.12582439 -0.07363078]
[ 0.0260968   0.12703272 -0.07397711]
[ 0.02652781  0.12583583 -0.07363406]
[ 0.02610088  0.12626253 -0.07375636]
[ 0.02625308  0.12583988 -0.07363522]
[ 0.02610233  0.12599055 -0.07367841]
[ 0.02615607  0.1258413  -0.07363563]
[ 0.02610284  0.12589451 -0.07365088]
[ 0.02612181  0.12584181 -0.07363578]
[ 0.02610301  0.1258606  -0.07364116]
[ 0.02610972  0.12584198 -0.07363583]
[ 0.02610308  0.12584862 -0.07363773]
[ 0.02610545  0.12584205 -0.07363585]
[ 0.0261031   0.12584439 -0.07363652]
[ 0.02610394  0.12584207 -0.07363585]
[ 0.02610311  0.1258429  -0.07363609]
[ 0.0261034   0.12584208 -0.07363585]
[ 0.02610311  0.12584237 -0.07363594]
[ 0.02610322  0.12584208 -0.07363586]
[ 0.02610311  0.12584218 -0.07363589]
[ 0.02610315  0.12584208 -0.07363586]
```

```
[ 0.02610311  0.12584212 -0.07363587]
[ 0.02610313  0.12584208 -0.07363586]
[ 0.02610311  0.12584209 -0.07363586]
[ 0.02610312  0.12584208 -0.07363586]
[ 0.02610311  0.12584209 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
```

```
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
```

```
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
[ 0.02610311  0.12584208 -0.07363586]
```

These are the opinions - note some decimal places are missing.

1.1 The limit matrix if there were no agendas:

[10]:
```python
A = np.array([
  [0, 0.5, 0.5],
  [1, 0, 0],
  [1, 0, 0],
], ndmin = 2)

# the bot fractions
c = np.array([0, 0, 0], ndmin = 2).T
A_tilde = np.concatenate((np.concatenate((A, c), axis = 1), # A c
    np.concatenate((np.zeros((1, 3)), np.array([1], ndmin = 2)), axis = 1)), #␣
 ↪0 1
    axis = 0)

# the strategic agent fractions
```

```
B = np.array([0, 0, 0], ndmin = 2).T
B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)

A_tilde_prime = np.concatenate((np.concatenate((A_tilde, B_tilde), axis = 1), #
 ↪A c
                        np.concatenate((np.zeros((1, 4)), np.array([1], ndmin =
 ↪2)), axis = 1)), # 0 1
                        axis = 0)

print(A_tilde_prime)
print()
print(np.linalg.matrix_power(A_tilde_prime, 2))
print(np.linalg.matrix_power(A_tilde_prime, 3))
print(np.linalg.matrix_power(A_tilde_prime, 4))
```

```
[[0.  0.5 0.5 0.  0. ]
 [1.  0.  0.  0.  0. ]
 [1.  0.  0.  0.  0. ]
 [0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  1. ]]

[[1.  0.  0.  0.  0. ]
 [0.  0.5 0.5 0.  0. ]
 [0.  0.5 0.5 0.  0. ]
 [0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  1. ]]
[[0.  0.5 0.5 0.  0. ]
 [1.  0.  0.  0.  0. ]
 [1.  0.  0.  0.  0. ]
 [0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  1. ]]
[[1.  0.  0.  0.  0. ]
 [0.  0.5 0.5 0.  0. ]
 [0.  0.5 0.5 0.  0. ]
 [0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  1. ]]
```

[11]:
```
print(np.linalg.matrix_power(A_tilde_prime, 1000000000000))
```

```
[[1.  0.  0.  0.  0. ]
 [0.  0.5 0.5 0.  0. ]
 [0.  0.5 0.5 0.  0. ]
 [0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  1. ]]
```

[12]:
```
print(np.linalg.matrix_power(A_tilde_prime, 1000000000001))
```

```
[[0.  0.5 0.5 0.  0. ]
 [1.  0.  0.  0.  0. ]
 [1.  0.  0.  0.  0. ]
 [0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  1. ]]
```