BotInfluence

August 5, 2021

1 Experimentation with Strategic Influence Network Model, Part 5a

1.0.1 More statics for the bot case + verification

James Yu

4 August 2021

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

[2]:
```
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, 6)), 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)
    res_mat = np.linalg.matrix_power(A_tilde_prime, 1000000000000)
    res_vec = res_mat @ w_0_prime
    return res_mat, res_vec
```

[3]:
```
def optimal_K_dynamic(A, c, B, z, delta = 0.8, tol = 10**(-18)):
    x = np.array([0, 0, 0, 0, 0], ndmin = 2).T
    A_tilde = np.concatenate((np.concatenate((A, c), axis = 1), # A c
        np.concatenate((np.zeros((1, 5)), np.array([1], ndmin = 2)), axis =␣
 ↪1)), # 0 1
        axis = 0)
    B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)
    w_0 = np.concatenate((x, np.array([z], ndmin = 2)), axis = 0)
    Q = 0.2 * np.identity(5)
    Q_tilde = 0.2 * np.identity(6)
    Q_tilde[5, :] = 0

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

    # first compute the sequence of optimal K_t matrices
    K = np.zeros((6, 6))
```

```
    K_t = [Q_tilde, K]
    K = Q_tilde
    current_difference = np.inf
    while abs(current_difference) > tol: # to avoid floating point error, don't converge all the way to zero (this is standard)
        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[:5, :5]
    c_nplus1 = np.array(expr[:5, 5], 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
```

1.0.2 Observation 2.1: Two agents with (pairwise) identical weights on everyone have identical limit opinions

[4]:
```
A = np.array([
      [0.1, 0.1, 0.1, 0.1, 0.1],
      [0.1, 0.1, 0.1, 0.1, 0.1],
      [0.2, 0.2, 0.1, 0.05, 0.35],
      [0.2, 0.2, 0.1, 0.05, 0.35],
      [0.2, 0.2, 0.1, 0.05, 0.35],
    ], ndmin = 2)
c = np.array([0.3, 0.3, 0.1, 0.1, 0.1,], ndmin = 2).T
B = np.array([0.2, 0.2, 0, 0, 0,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.        0.        0.        0.        0.        0.6428571 0.3571429]
 [0.        0.        0.        0.        0.        0.6428571 0.3571429]
 [0.        0.        0.        0.        0.        0.7142857 0.2857143]
 [0.        0.        0.        0.        0.        0.7142857 0.2857143]
 [0.        0.        0.        0.        0.        0.7142857 0.2857143]
 [0.        0.        0.        0.        0.        1.        0.       ]
 [0.        0.        0.        0.        0.        0.        1.       ]]
```

[5]:
```
vec
```

[5]:
```
array([[-0.09756098],
       [-0.09756098],
       [ 0.12195122],
       [ 0.12195122],
       [ 0.12195122],
       [ 1.        ],
       [-2.07317073]])
```

Agents 1 and 2 are identical in their a_{ik}, b_i entries, as are agents 3, 4 and 5 together. According to the above, they have the same limit opinions and influence shares.

1.0.3 Observation 2.2: Identical diagonal and off-diagonal implies the bot and strategic agent have equal weighting

[6]:
```
A = np.array([
      [0, 1/6, 1/6, 1/6, 1/6],
      [1/6, 0, 1/6, 1/6, 1/6],
      [1/6, 1/6, 0, 1/6, 1/6],
      [1/6, 1/6, 1/6, 0, 1/6],
      [1/6, 1/6, 1/6, 1/6, 0],
    ], ndmin = 2)
c = np.array([1/6, 1/6, 1/6, 1/6, 1/6], ndmin = 2).T
B = np.array([1/6, 1/6, 1/6, 1/6, 1/6], ndmin = 2).T
```

```
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

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

[7]:
```
A = np.array([
      [0, 1/6, 1/6, 1/6, 1/6],
      [1/7, 1/7, 1/7, 1/7, 1/7],
      [1/6, 1/6, 0, 1/6, 1/6],
      [1/6, 1/6, 1/6, 0, 1/6],
      [1/6, 1/6, 1/6, 1/6, 0],
    ], ndmin = 2)
c = np.array([1/6, 1/7, 1/6, 1/6, 1/6], ndmin = 2).T
B = np.array([1/6, 1/7, 1/6, 1/6, 1/6], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

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

[8]:
```
A = np.array([
      [0, 0.05, 0.1, 0.05, 0],
      [0.05, 0, 0.05, 0.05, 0.05],
      [0, 0, 0, 0.1, 0.1],
      [0, 0.1, 0.05, 0.05, 0],
      [0, 0.1, 0.1, 0, 0],
    ], ndmin = 2)
c = np.array([0.4, 0.4, 0.4, 0.4, 0.4], ndmin = 2).T
B = np.array([0.4, 0.4, 0.4, 0.4, 0.4], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

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

[9]:
```
A = np.array([
      [0, 0.05, 0.1, 0.05, 0],
      [0.05, 0.2, 0.25, 0.05, 0.05],
      [0.2, 0, 0.2, 0.1, 0.1],
      [0, 0.1, 0.05, 0.05, 0],
      [0, 0.1, 0.1, 0, 0],
    ], ndmin = 2)
c = np.array([0.4, 0.2, 0.2, 0.4, 0.4], ndmin = 2).T
B = np.array([0.4, 0.2, 0.2, 0.4, 0.4], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

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

[10]:
```
A = np.array([
      [0, 0.05, 0.1, 0.05, 0],
      [0.05, 0.2, 0.25, 0.05, 0.05],
      [0.2, 0, 0.2, 0.1, 0.1],
      [0, 0.1, 0.05, 0.05, 0],
      [0, 0.1, 0.1, 0, 0],
    ], ndmin = 2)
c = np.array([0.4, 0.3, 0.2, 0.4, 0.4], ndmin = 2).T
B = np.array([0.4, 0.1, 0.2, 0.4, 0.4], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.        0.        0.        0.        0.        0.5076669 0.4923331]
 [0.        0.        0.        0.        0.        0.6288468 0.3711532]
 [0.        0.        0.        0.        0.        0.5053242 0.4946758]
 [0.        0.        0.        0.        0.        0.513843  0.486157 ]
 [0.        0.        0.        0.        0.        0.5134171 0.4865829]
```

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

[11]:
```
A = np.array([
      [0, 1/6, 1/6, 1/6, 1/6],
      [1/6, 0, 1/6, 1/6, 1/6],
      [1/6, 1/6, 0, 1/6, 1/6],
      [1/6, 1/6, 1/6, 0, 1/6],
      [1/6, 1/6, 1/6, 1/6, 0],
    ], ndmin = 2)
c = np.array([1/6, 1/12, 1/6, 1/6, 1/6], ndmin = 2).T
B = np.array([1/6, 3/12, 1/6, 1/6, 1/6], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.        0.        0.        0.        0.        0.4642857 0.5357143]
 [0.        0.        0.        0.        0.        0.3928571 0.6071429]
 [0.        0.        0.        0.        0.        0.4642857 0.5357143]
 [0.        0.        0.        0.        0.        0.4642857 0.5357143]
 [0.        0.        0.        0.        0.        0.4642857 0.5357143]
 [0.        0.        0.        0.        0.        1.        0.       ]
 [0.        0.        0.        0.        0.        0.        1.       ]]
```

[12]:
```
A = np.array([
      [1/7, 1/7, 1/7, 1/7, 1/7],
      [1/7, 1/7, 1/7, 1/7, 1/7],
      [1/7, 1/7, 1/7, 1/7, 1/7],
      [1/7, 1/7, 1/7, 1/7, 1/7],
      [1/7, 1/7, 1/7, 1/7, 1/7],
    ], ndmin = 2)
c = np.array([1/7, 3/14, 1/7, 1/7, 1/7], ndmin = 2).T
B = np.array([1/7, 1/14, 1/7, 1/7, 1/7], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.        0.        0.        0.        0.        0.5357143 0.4642857]
 [0.        0.        0.        0.        0.        0.6071429 0.3928571]
 [0.        0.        0.        0.        0.        0.5357143 0.4642857]
 [0.        0.        0.        0.        0.        0.5357143 0.4642857]
 [0.        0.        0.        0.        0.        0.5357143 0.4642857]
 [0.        0.        0.        0.        0.        1.        0.       ]
 [0.        0.        0.        0.        0.        0.        1.       ]]
```

Conclusion: this may be dependent exclusively on having the property of $b_i = c_i, \forall i$.
Exploring a special case:

[13]:
```
A = np.array([
      [0, 0.05, 0.1, 0.05, 0],
      [0.05, 0.2, 0.25, 0.05, 0.05],
      [0.2, 0, 0.2, 0.1, 0.1],
      [0, 0.1, 0.05, 0.05, 0],
      [0, 0.1, 0.1, 0, 0],
    ], ndmin = 2)
c = np.array([0.4, 0.2, 0.2, 0.4, 0.4], ndmin = 2).T
B = np.array([0.4, 0.2, 0.2, 0.4, 0.4], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1)
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

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

[14]:
```
A_tilde_prime = np.concatenate((np.concatenate((A_tilde_, B_tilde_), axis = 1),␣
 ↪# A c
                    np.concatenate((np.zeros((1, 6)), np.array([1], ndmin =␣
 ↪2)), axis = 1)), # 0 1
                    axis = 0)

np.linalg.matrix_power(A_tilde_prime, 2)
```

[14]:
```
array([[0.0225, 0.015 , 0.035 , 0.015 , 0.0125, 0.45  , 0.45  ],
       [0.06  , 0.0525, 0.1125, 0.04  , 0.035 , 0.35  , 0.35  ],
       [0.04  , 0.03  , 0.075 , 0.035 , 0.02  , 0.4   , 0.4   ],
       [0.015 , 0.025 , 0.0375, 0.0125, 0.01  , 0.45  , 0.45  ],
       [0.025 , 0.02  , 0.045 , 0.015 , 0.015 , 0.44  , 0.44  ],
       [0.    , 0.    , 0.    , 0.    , 0.    , 1.    , 0.    ],
       [0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 1.    ]])
```

[15]:
```
np.round(np.linalg.matrix_power(A_tilde_prime, 3), 5)
```

[15]:
```
array([[0.00775, 0.00688, 0.015  , 0.00613, 0.00425, 0.48   , 0.48   ],
       [0.02513, 0.021  , 0.04713, 0.01888, 0.01388, 0.437  , 0.437  ],
       [0.0165 , 0.0135 , 0.03025, 0.01275, 0.009  , 0.459  , 0.459  ],
       [0.00875, 0.008  , 0.01688, 0.00638, 0.005  , 0.4775 , 0.4775 ],
       [0.01   , 0.00825, 0.01875, 0.0075 , 0.0055 , 0.475  , 0.475  ],
       [0.     , 0.     , 0.     , 0.     , 0.     , 1.     , 0.     ],
       [0.     , 0.     , 0.     , 0.     , 0.     , 0.     , 1.     ]])
```

If we have any matrix A, and two columns in B are identical, they must be identical in AB because during the matrix multiplication, the same columns are being used to construct each cell

in the corresponding column of AB. Notably, if we have column vector $\vec{B}_i$ in column i and column vector $\vec{B}_j$ in column j such that $\vec{B}_i = \vec{B}_j$, then in AB, we must have the vector $\{\vec{A}_1 \cdot \vec{B}_i, \vec{A}_2 \cdot \vec{B}_i, \vec{A}_3 \cdot \vec{B}_i, \ldots \vec{A}_n \cdot \vec{B}_i\}$ in column i and $\{\vec{A}_1 \cdot \vec{B}_j, \vec{A}_2 \cdot \vec{B}_j, \vec{A}_3 \cdot \vec{B}_j, \ldots \vec{A}_n \cdot \vec{B}_j\}$ in column j. Since $\vec{B}_i = \vec{B}_j$, it follows that $A\vec{B}_i = A\vec{B}_j$.

By induction, this means that they are identical at the limit of A^t, so as long as the initial weights are the same, the long-run weights are the same.

1.0.4 Observation 2.3a: Given targeted influence (one unique naive agent per strategic agent), the strategic agent always has more influence toward the agent listening to her than the bot, as does the bot in the converse case

[16]:
```
A = np.array([
      [0.2, 0.3, 0.05, 0.05, 0.3],
      [0.2, 0.3, 0.05, 0.05, 0.3],
      [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
      [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
      [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
    ], ndmin = 2)
c = np.array([0, 0.1, 0, 0, 0,], ndmin = 2).T
B = np.array([0.1, 0, 0, 0, 0,], ndmin = 2).T
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1, tol =␣
 ↪10**(-16))
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.        0.        0.        0.        0.        0.4332246 0.5667754]
 [0.        0.        0.        0.        0.        0.5332246 0.4667754]
 [0.        0.        0.        0.        0.        0.4682634 0.5317366]
 [0.        0.        0.        0.        0.        0.4640828 0.5359172]
 [0.        0.        0.        0.        0.        0.4666499 0.5333501]
 [0.        0.        0.        0.        0.        1.        0.       ]
 [0.        0.        0.        0.        0.        0.        1.       ]]
```

In this example, this is the case (and should always be the case).

1.0.5 Observation 2.3b: Given a subnetwork linked through agent i, all agents in the subnetwork share agent i's limit opinion

[17]:
```
A = np.array([
      [0.2, 0.3, 0, 0, 0],
      [0.2, 0.3, 0, 0, 0],
      [0.1285,   0,   0.3185 + 0.0907,   0.2507,   0.2116],
      [0.1975,   0,   0.2863 + 0.0629,   0.2396,   0.2137],
      [0.1256,   0,   0.0253 + 0.0711,   0.2244,   0.5536],
    ], ndmin = 2)
c = np.array([0, 0.5, 0, 0, 0,], ndmin = 2).T
B = np.array([0.5, 0, 0, 0, 0,], ndmin = 2).T
```

```
rs, A_tilde_, B_tilde_, w_0_, Ks, xs, ps = optimal_K_dynamic(A, c, B, 1, tol =␣
↪10**(-16))
mat, vec = l_matrix(rs[-1].item(), A_tilde_, B_tilde_, w_0_)
print(np.round(mat, 7))
```

```
[[0.  0.  0.  0.  0.  0.3 0.7]
 [0.  0.  0.  0.  0.  0.8 0.2]
 [0.  0.  0.  0.  0.  0.3 0.7]
 [0.  0.  0.  0.  0.  0.3 0.7]
 [0.  0.  0.  0.  0.  0.3 0.7]
 [0.  0.  0.  0.  0.  1.  0. ]
 [0.  0.  0.  0.  0.  0.  1. ]]
```

[18]:
```
vec
```

[18]:
```
array([[-0.06914163],
       [ 0.69453096],
       [-0.06914163],
       [-0.06914163],
       [-0.06914163],
       [ 1.        ],
       [-0.52734518]])
```

This is also the case, and the observation can be expanded to note that the shares of opinion $s_{ib}, s_{jb}, s_{is}, s_{js}$ are the same too