StrategicInfluence6

July 13, 2021

1 Experimentation with Strategic Influence Network Model, Part 6

1.0.1 Verification of steady-state vs optimal trajectories

James Yu
13 July 2021

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

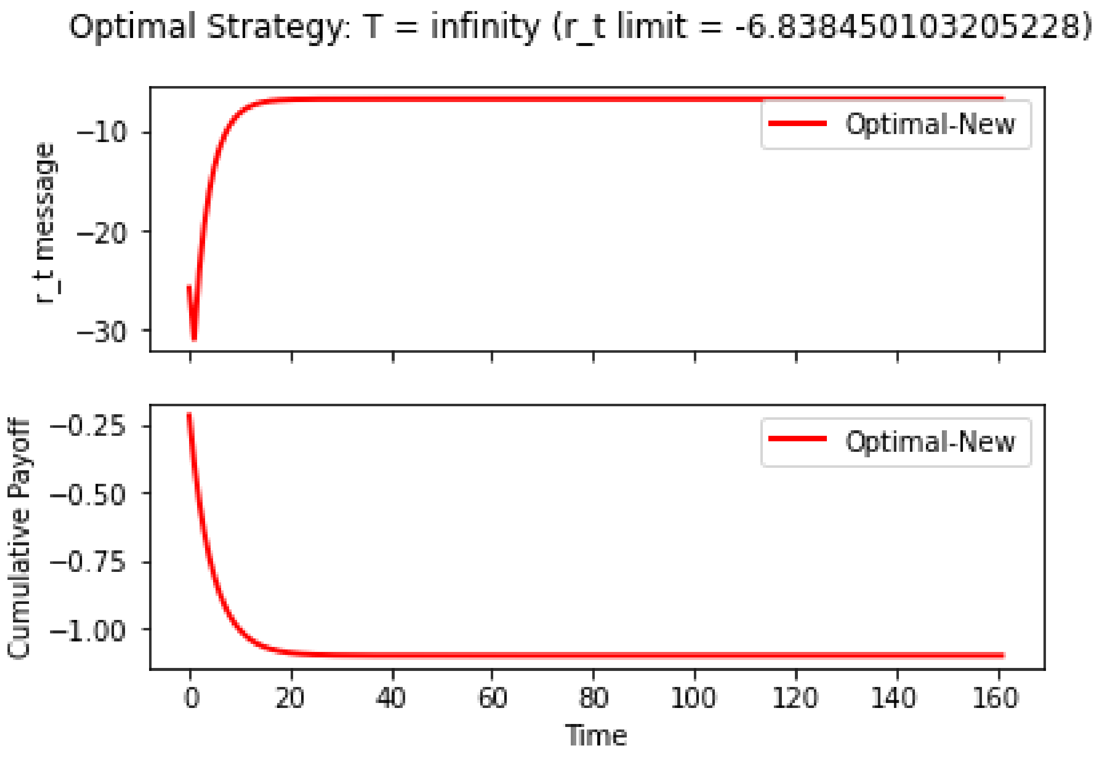
1.0.2 First, computation of necessary values:

[2]:
```
def optimal_K(z, delta = 0.8):
    A = np.array([
      [0.217,    0.2022,    0.2358,    0.1256,    0.1403],
      [0.8988*0.2497,    0.8988*0.0107,    0.8988*0.2334,    0.8988*0.1282,    0.
 ↪8988*0.378],
      [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, 0.1012, 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 = np.array([0.0791, 0, 0, 0, 0,], ndmin = 2).T
    B_tilde = np.concatenate((B, np.array([0], ndmin = 2)), axis = 0)
    x = np.array([-0.98, -4.62, 2.74, 4.67, 2.15,], ndmin = 2).T
    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) != 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[: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
```

[3]:
```
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()
```



1.1 The Comparison:

[4]:
```
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 = A_tilde_prime @ np.concatenate((w_0_, np.array([rs[0].item()],␣
 ↪ndmin = 2)), axis = 0)
w_0_alt = A_tilde_prime @ np.concatenate((w_0_, np.array([rs[-1].item()], ndmin␣
 ↪= 2)), axis = 0)
```

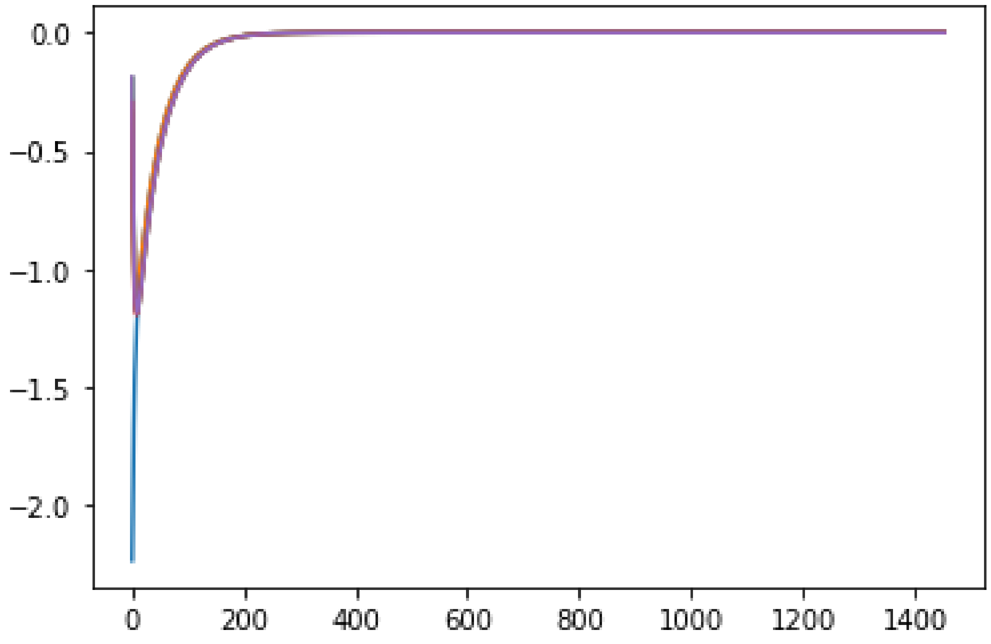
```python
print("ACTUAL (using r_t):")
print(w_0_prime)
print("ESTIMATE (using r_ss):")
print(w_0_alt)
diffs = []
i = 0
for r in rs[1:]:
    w_0_prime[6] = r.item()
    w_0_prime = A_tilde_prime @ w_0_prime
    w_0_alt = A_tilde_prime @ w_0_alt
    diff = w_0_prime - w_0_alt
    diffs.append(diff)
    if i < 5 or i > 155:
        print()
        print(f"ACTUAL (using r_t, t = {i}):")
        print(w_0_prime)
        print("ESTIMATE (using r_ss):")
        print(w_0_alt)
        print("DIFFERENCE:")
        print(diff)
    i += 1

while np.any(np.round(w_0_prime - w_0_alt, 14)):
    w_0_prime = A_tilde_prime @ w_0_prime # by this point, the strategic agent␣
 ↪has reached r_ss
    w_0_alt = A_tilde_prime @ w_0_alt
    diff = w_0_prime - w_0_alt
    diffs.append(diff)
    i += 1
    if i % 100 == 0:
        print(np.round(w_0_prime - w_0_alt, 14))

print(len(diffs), "total iterations needed to match.")
```

```
ACTUAL (using r_t):
[[ -1.65509311]
 [  2.59098488]
 [  1.953435  ]
 [  1.878701  ]
 [  1.85594   ]
 [ 10.        ]
 [-25.82247922]]
ESTIMATE (using r_ss):
[[-0.1534564 ]
 [ 2.59098488]
 [ 1.953435  ]
 [ 1.878701  ]
 [ 1.85594   ]
 [10.        ]
 [-6.8384501 ]]

ACTUAL (using r_t, t = 0):
[[ -1.32559258]
 [  1.92228065]
 [  1.50819916]
 [  1.24211164]
 [  1.47479012]
 [ 10.        ]
 [-30.93941492]]
ESTIMATE (using r_ss):
[[ 0.9066489 ]
 [ 2.25929351]
 [ 1.70115947]
 [ 1.53868489]
 [ 1.66339569]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[ -2.23224148]
 [ -0.33701287]
 [ -0.19296032]
 [ -0.29657325]
 [ -0.18860557]
 [  0.        ]
 [-24.10096482]]

ACTUAL (using r_t, t = 1):
[[ -1.0810707 ]
 [  1.69355195]
 [  1.10783662]
 [  0.90367693]
 [  1.10351083]
 [ 10.        ]
 [-24.02854486]]
ESTIMATE (using r_ss):
[[ 0.9404172 ]
 [ 2.33650607]
 [ 1.60096443]
 [ 1.53235124]
 [ 1.58368695]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[ -2.0214879 ]
 [ -0.64295413]
 [ -0.49312781]
 [ -0.6286743 ]
 [ -0.48017612]
 [  0.        ]
 [-17.19009475]]

ACTUAL (using r_t, t = 2):
[[ -0.90241527]
 [  1.497105  ]
 [  0.82758824]
 [  0.66252783]
 [  0.82634603]
 [ 10.        ]
 [-19.46664216]]
ESTIMATE (using r_ss):
[[ 0.92775266]
 [ 2.2959978 ]
 [ 1.56194049]
 [ 1.49664   ]
 [ 1.54533509]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[ -1.83016793]
 [ -0.7988928 ]
 [ -0.73435226]
 [ -0.83411217]
 [ -0.71898907]
 [  0.        ]
 [-12.62819206]]

ACTUAL (using r_t, t = 3):
[[ -0.7714769 ]
 [  1.35456854]
 [  0.62436446]
 [  0.48821122]
 [  0.6201752 ]
 [ 10.        ]
 [-16.08928656]]
ESTIMATE (using r_ss):
[[ 0.89774575]
 [ 2.26743471]
 [ 1.52714182]
 [ 1.46366603]
 [ 1.5106318 ]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
```

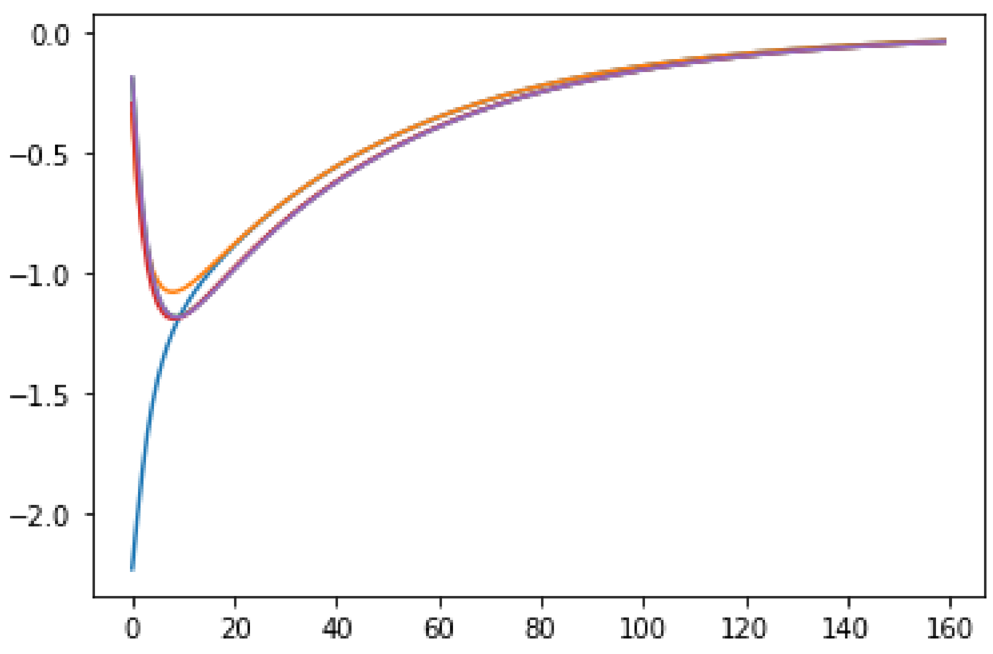
```
[[-1.66922264]
 [-0.91286617]
 [-0.90277736]
 [-0.97545481]
 [-0.8904566 ]
 [ 0.        ]
 [-9.25083646]]

ACTUAL (using r_t, t = 4):
[[ -0.67546433]
 [  1.24982039]
 [  0.47620829]
 [  0.36109807]
 [  0.46809233]
 [ 10.        ]
 [-13.62203104]]
ESTIMATE (using r_ss):
[[ 0.86824286]
 [ 2.2375357 ]
 [ 1.49400209]
 [ 1.43066353]
 [ 1.47734058]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[-1.54370719]
 [-0.98771532]
 [-1.0177938 ]
 [-1.06956546]
 [-1.00924825]
 [ 0.        ]
 [-6.78358093]]

ACTUAL (using r_t, t = 156):
[[-0.41107847]
 [ 0.96122649]
 [ 0.06976123]
 [ 0.01239086]
 [ 0.04761921]
 [10.        ]
 [-6.8384501 ]]
ESTIMATE (using r_ss):
[[-0.3722265 ]
 [ 0.99998781]
 [ 0.11301405]
 [ 0.05546314]
 [ 0.09103818]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[-0.03885197]
 [-0.03876132]
 [-0.04325282]
 [-0.04307228]
 [-0.04341896]
 [ 0.        ]
 [ 0.        ]]

ACTUAL (using r_t, t = 157):
[[-0.41107847]
 [ 0.96122649]
 [ 0.06976123]
 [ 0.01239086]
 [ 0.04761921]
 [10.        ]
 [-6.8384501 ]]
ESTIMATE (using r_ss):
[[-0.37310948]
 [ 0.99910689]
 [ 0.11203105]
 [ 0.05448425]
 [ 0.0900514 ]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[-0.03796899]
 [-0.0378804 ]
 [-0.04226983]
 [-0.04209338]
 [-0.04243219]
 [ 0.        ]
 [ 0.        ]]

ACTUAL (using r_t, t = 158):
[[-0.41107847]
 [ 0.96122649]
 [ 0.06976123]
 [ 0.01239086]
 [ 0.04761921]
 [10.        ]
 [-6.8384501 ]]
ESTIMATE (using r_ss):
[[-0.37397239]
 [ 0.99824599]
 [ 0.11107039]
 [ 0.0535276 ]
 [ 0.08908706]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[-0.03710608]
 [-0.0370195 ]
 [-0.04130917]
 [-0.04113674]
 [-0.04146784]
 [ 0.        ]
 [ 0.        ]]

ACTUAL (using r_t, t = 159):
[[-0.41107847]
 [ 0.96122649]
 [ 0.06976123]
 [ 0.01239086]
 [ 0.04761921]
 [10.        ]
 [-6.8384501 ]]
ESTIMATE (using r_ss):
[[-0.37481569]
 [ 0.99740466]
 [ 0.11013157]
 [ 0.0525927 ]
 [ 0.08814463]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[-0.03626278]
 [-0.03617817]
 [-0.04037035]
 [-0.04020183]
 [-0.04052541]
 [ 0.        ]
 [ 0.        ]]

ACTUAL (using r_t, t = 160):
[[-0.41107847]
 [ 0.96122649]
 [ 0.06976123]
 [ 0.01239086]
 [ 0.04761921]
 [10.        ]
 [-6.8384501 ]]
ESTIMATE (using r_ss):
[[-0.37563983]
 [ 0.99658245]
 [ 0.10921408]
 [ 0.05167904]
 [ 0.08722362]
 [10.        ]
 [-6.8384501 ]]
DIFFERENCE:
[[-0.03543864]
 [-0.03535595]
 [-0.03945286]
 [-0.03928817]
 [-0.0396044 ]
 [ 0.        ]
 [ 0.        ]]
[[-0.01445777]
 [-0.01442404]
 [-0.01609543]
 [-0.01602825]
 [-0.01615726]
 [ 0.        ]
 [ 0.        ]]
[[-0.00145112]
 [-0.00144773]
 [-0.00161549]
 [-0.00160874]
 [-0.00162169]
 [ 0.        ]
 [ 0.        ]]
[[-0.00014565]
 [-0.00014531]
 [-0.00016215]
 [-0.00016147]
 [-0.00016277]
 [ 0.        ]
 [ 0.        ]]
[[-1.46185063e-05]
 [-1.45843976e-05]
 [-1.62743787e-05]
 [-1.62064464e-05]
 [-1.63368908e-05]
 [ 0.00000000e+00]
 [ 0.00000000e+00]]
[[-1.46724816e-06]
 [-1.46382469e-06]
 [-1.63344679e-06]
 [-1.62662848e-06]
 [-1.63972108e-06]
 [ 0.00000000e+00]
 [ 0.00000000e+00]]
```

```
[[-1.4726656e-07]
 [-1.4692295e-07]
 [-1.6394779e-07]
 [-1.6326344e-07]
 [-1.6457753e-07]
 [ 0.0000000e+00]
 [ 0.0000000e+00]]
[[-1.478103e-08]
 [-1.474654e-08]
 [-1.645531e-08]
 [-1.638663e-08]
 [-1.651852e-08]
 [ 0.000000e+00]
 [ 0.000000e+00]]
[[-1.48356e-09]
 [-1.48010e-09]
 [-1.65161e-09]
 [-1.64471e-09]
 [-1.65795e-09]
 [ 0.00000e+00]
 [ 0.00000e+00]]
[[-1.4890e-10]
 [-1.4856e-10]
 [-1.6577e-10]
 [-1.6508e-10]
 [-1.6641e-10]
 [ 0.0000e+00]
 [ 0.0000e+00]]
[[-1.495e-11]
 [-1.491e-11]
 [-1.664e-11]
 [-1.657e-11]
 [-1.670e-11]
 [ 0.000e+00]
 [ 0.000e+00]]
[[-1.50e-12]
 [-1.50e-12]
 [-1.67e-12]
 [-1.66e-12]
 [-1.68e-12]
 [ 0.00e+00]
 [ 0.00e+00]]
[[-1.5e-13]
 [-1.5e-13]
 [-1.7e-13]
 [-1.7e-13]
 [-1.7e-13]
 [ 0.0e+00]
 [ 0.0e+00]]
[[-2.e-14]
 [-2.e-14]
 [-2.e-14]
 [-2.e-14]
 [-2.e-14]
 [ 0.e+00]
 [ 0.e+00]]
1454 total iterations needed to match.
```

[5]:
```
for i in range(5):
    plt.plot(range(len(diffs)), [d[i] for d in diffs])
plt.show()
```



[6]:
```
for i in range(5):
    plt.plot(range(160), [d[i] for d in diffs[:160]])
plt.show()
```



These are the differences between the r_{ss} estimate and the actual result for each naive agent (the blue line is agent 1, orange line is agent 2, and the remaining lines are the remaining agents).