Split_Influence_Multi_Agent_Case

September 16, 2021

1 Benchmarks of Competitive Influence Model (revised)

James Yu, 16 September 2021

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

[2]:
```
def M(K, B, R, L, delta):
    """Computes M_{t-1} given B_l \forall l, K_t^l \forall l,
        R_l \forall l, number of strategic agents L, and delta."""
    # handle the generic structure first:
    template = [(B[l].T @ K[l] @ B[l]).item() for l in range(L)]
    base = [template.copy() for l_prime in range(L)]
    # then change the diagonals to construct M_{t-1}:
    for l in range(L): base[l][l] = (B[l].T @ K[l] @ B[l] + R[l]/delta).item()
    return np.array(base, ndmin = 2)

def H(B, K, A, L):
    """Computes H_{t-1} given B_l \forall l, K_t^l \forall l,
        A, and number of strategic agents L."""
    return np.concatenate(tuple(B[l].T @ K[l] @ A for l in range(L)), axis = 0)

def C_l(B, K, k, h, L, c, x, n):
    """Computes C_{t-1}^h (displayed as C_{t-1}^l) given B_l \forall l, K_t^l␣
↪\forall l,
        k_t^l \forall l, a specific naive agent h, number of strategic agents␣
↪L,
        c_l \forall l, x_l \forall l, and number of naive agents n"""
    return np.concatenate(tuple(B[l].T @ K[l] @ A @ ((x[h] - x[l]) * np.
↪ones((n, 1)))
                            + B[l].T @ K[l] @ c[l]
                            + 0.5 * B[l].T @ k[l].T for l in range(L)), axis = 0)

def E(M_, H_):
    """Computes the generic E_{t-1} given M_{t-1} and H_{t-1}."""
    return np.linalg.inv(M_) @ H_
```

```python
def F(M_, C_l_, l):
    """Computes F_{t-1}^l given M_{t-1}, C_{t-1}^l, and specific naive agent l.
 ↪"""
    return (np.linalg.inv(M_) @ C_l_)[l:l+1, :]

def G(A, B, E_, L):
    """Computes the generic G_{t-1} given A, B_l \forall l,
        E_{t-1}, and number of strategic agents L."""
    return A - sum([B[l] @ E_[l:l+1, :] for l in range(L)])

def g_l(B, E_, h, x, F_, L):
    """Computes g_{t-1}^l given B_l \forall l, E_{t-1}^l,
        a particular naive agent h, x_l \forall l, F_{t-1}^l \forall l,
        number of strategic agents L, number of naive agents n, and c_h."""
    return - sum([B[l] @ (E_[l:l+1, :] @ ((x[h] - x[l]) * np.ones((n, 1))) +␣
 ↪F_[l]) for l in range(L)]) + c[h]
```

[3]:
```python
def K_t_minus_1(Q, K, E_, R, G_, L, delta):
    return [Q[l] + E_[l:l+1, :].T @ R[l] @ E_[l:l+1, :]
            + delta * G_.T @ K[l] @ G_ for l in range(L)]

def k_t_minus_1(K, k, G_, g, E_, F_, R, L, delta):
    return [2*delta* g[l].T @ K[l] @ G_ + delta * k[l] @ G_
            + 2 * F_[l].T @ R[l] @ E_[l:l+1, :] for l in range(L)]

def kappa_t_minus_1(K, k, kappa, g_, F_, R, L, delta):
    return [-delta * (g_[l].T @ K[l] @ g_[l] + k[l] @ g_[l] - kappa[l])
            - (F_[l].T @ R[l] @ F_[l]) for l in range(L)]
```

[4]:
```python
def solve(K_t, k_t, kappa_t, A, B, delta, n, m, L, Q, R, x, c, tol = 300):
    historical_K = [K_t]
    historical_k = [k_t]
    historical_kappa = [kappa_t]
    max_distances = defaultdict(list)
    counter = 0
    while True:
        M_ = M(K_t, B, R, L, delta)
        H_ = H(B, K_t, A, L)
        E_ = E(M_, H_)
        G_ = G(A, B, E_, L)
        K_new = K_t_minus_1(Q, K_t, E_, R, G_, L, delta)
        F_ = [F(M_, C_l(B, K_t, k_t, l, L, c, x, n), l) for l in range(L)]
        g = [g_l(B, E_, h, x, F_, L) for h in range(L)]
        k_new = k_t_minus_1(K_t, k_t, G_, g, E_, F_, R, L, delta)
        kappa_new = kappa_t_minus_1(K_t, k_t, kappa_t, g, F_, R, L, delta)
        cd_K = [np.max(np.abs(K_t[l] - K_new[l])) for l in range(L)]
        cd_k = [np.max(np.abs(k_t[l] - k_new[l])) for l in range(L)]
```

```
        cd_kappa = [np.max(np.abs(kappa_t[l] - kappa_new[l])) for l in range(L)]
        K_t = K_new
        k_t = k_new
        kappa_t = kappa_new
        historical_K.insert(0, K_t)
        historical_k.insert(0, k_t)
        historical_kappa.insert(0, kappa_t)
        for l in range(L):
            max_distances[(l+1, "K")].append(cd_K[l])
            max_distances[(l+1, "k")].append(cd_k[l])
            max_distances[(l+1, "kappa")].append(cd_kappa[l])
        counter += 1
        if sum(cd_K + cd_k + cd_kappa) == 0 or counter > tol:
            return max_distances, historical_K, historical_k, historical_kappa
```

[5]:
```
def converge_plot(max_distances, tol = 300):
    fig, ax = plt.subplots()
    fig.suptitle(f"Convergence to Zero over Time ({len(max_distances[(1, 'K')]) + 1} iterations needed {['', '- rounding error observed'][len(max_distances[(1, 'K')]) + 1 == tol + 2]})")
    for l in max_distances:
        ax.plot(range(len(max_distances[l])), max_distances[l], label = f"Agent {l[0]}: {l[1]}")
    plt.xlabel("Time (iterations)")
    plt.ylabel("Maximum distance of differences from 0")
    ax.legend()
    plt.show()
```

[6]:
```
def optimal(X_init, historical_K, historical_k, historical_kappa, infinite = True):
    X_t = [a.copy() for a in X_init]
    xs = defaultdict(list)
    for l in range(L):
        xs[l].append(X_t[l])

    rs = defaultdict(list)
    payoffs = defaultdict(list)
    payoff = defaultdict(lambda: 0)
    i = 0
    while [i < len(historical_K), True][infinite]:
        K_t = historical_K[[i, 0][infinite]]
        k_t = historical_k[[i, 0][infinite]]
        M_ = M(K_t, B, R, L, delta)
        H_ = H(B, K_t, A, L)
        E_ = E(M_, H_)
        G_ = G(A, B, E_, L)
        F_ = [F(M_, C_l(B, K_t, k_t, l, L, c, x, n), l) for l in range(L)]
```

```
        g = [g_l(B, E_, h, x, F_, L) for h in range(L)]
        for l in range(L):
            Y_new = -1 * E_[l:l+1, :] @ X_t[l] - F(M_, C_l(B, K_t, k_t, l, L,␣
↪c, x, n), l)
            rs[l].append(Y_new)
            payoff[l] += (-1 * delta**i * (X_t[l].T @ Q[l] @ X_t[l])).item() +␣
↪(-1 * delta**i * (Y_new.T @ R[l] @ Y_new)).item()
            payoffs[l].append(payoff[l])
            X_new = G_ @ X_t[l] + g[l]
            xs[l].append(X_new)
            if infinite == True and np.max(X_t[l] - X_new) == 0:
                return xs, rs, payoffs
            X_t[l] = X_new
        i += 1

    return xs, rs, payoffs
```

[7]:
```
def do_plot(rs, r, payoffs, set_cap = np.inf):
    fig, sub = plt.subplots(2, sharex=True)
    fig.suptitle(f"Terminal Strategy: {', '.join(['$r_{ss}^' + str(l+1) + '$ =␣
↪' + str(round(rs[l][:min(len(rs[l]), set_cap)][-1].item() + r[l], 2)) for l␣
↪in range(L)])}")

    for l in range(L):
        sub[0].plot(range(min(len(rs[l]), set_cap)), [a.item() + r[l] for a in␣
↪rs[l][:min(len(rs[l]), set_cap)]], label = f"Optimal: Agent {l+1}")
    sub[0].set(ylabel = "r_t message")

    for l in range(L):
        sub[1].plot(range(min(len(payoffs[l]), set_cap)), payoffs[l][:
↪min(len(payoffs[l]), set_cap)], label = f"Optimal: Agent {l+1}")
    sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")

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

1.1 1-naive 1-strategic benchmark case

(that is, one naive agent and one strategic agent in this particular model)

[8]:
```
A = np.array([
   [0.7],
], ndmin = 2)

B_1 = np.array([
   0.3
```

```
], ndmin = 2).T

B = [B_1]

delta = 0.8
n = 1
m = 1
L = 1
Q = [1 * np.identity(n)]
R = [0.2 * np.identity(m)] # COST

x = [0]
r = [0]
c_base = sum([B[l] @ np.array([[r[l]]], ndmin = 2) for l in range(L)])
c = [c_base + (A - np.identity(n)) @ (x[l] * np.ones((n, 1))) for l in range(L)]

X_0_1 = np.array([
  4
], ndmin = 2).T # starting with an X_0 > 0
X_0 = [X_0_1]
```

[9]:

```
max_distances, historical_K, historical_k, historical_kappa = solve([Q[0]], [np.
 ↪zeros((1, n))], [0], A, B, delta, n, m, L, Q, R, x, c)
converge_plot(max_distances)
```

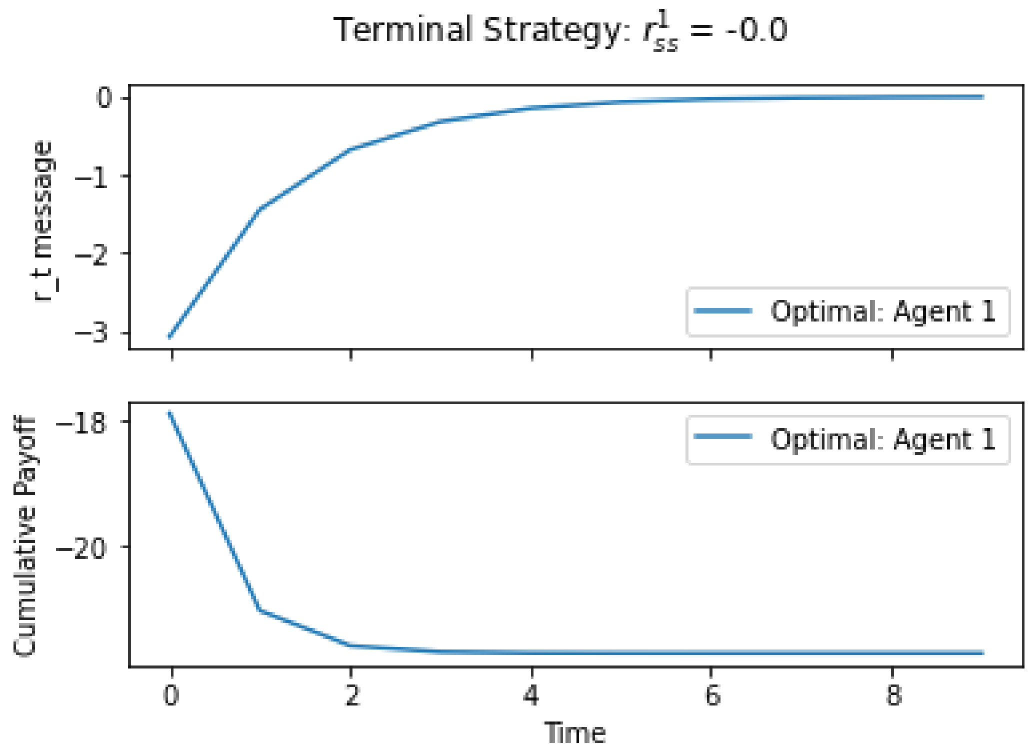
[10]:
```
historical_K
```

[10]:
```
[[array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744267]])],
 [array([[1.35744266]])],
 [array([[1.3574426]])],
 [array([[1.35744229]])],
 [array([[1.35744051]])],
 [array([[1.35743044]])],
 [array([[1.35737352]])],
 [array([[1.35705176]])],
 [array([[1.35523386]])],
 [array([[1.34499277]])],
 [array([[1.28823529]])],
 [array([[1.]])]]
```

To clarify: this plot says convergence of K gets close to zero sometime just after 5 iterations (and due to computer precision, takes 23 iterations to get exactly to zero). I looked at some comparable setups in QuantEcon (Prof. Jesse Perla's lectures) and convergence is best represented by "close to zero" rather than exactly at zero on computers.

k, κ are zero immediately as expected.

[11]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, set_cap = 10)
```



Here we can more clearly see a convergence at around 6-8 iterations.

[12]:
```
payoffs[0][-1]
```

[12]: -21.719082733041443

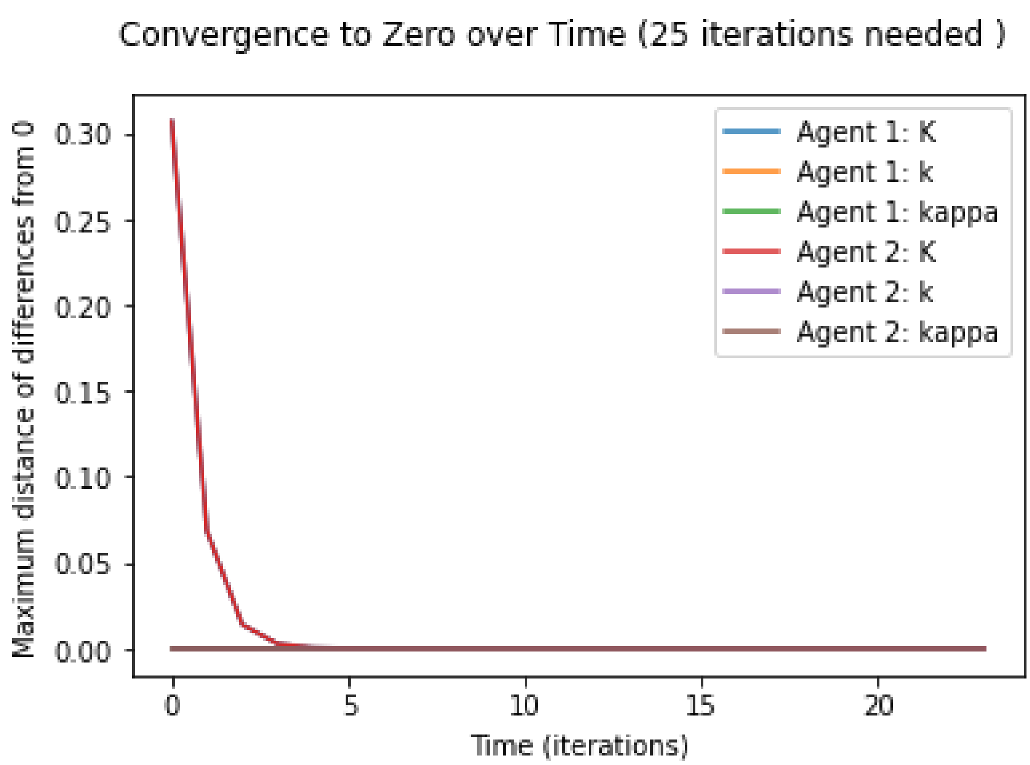
Cumulative payoff is -21.7.

1.2 1-naive 2-strategic case

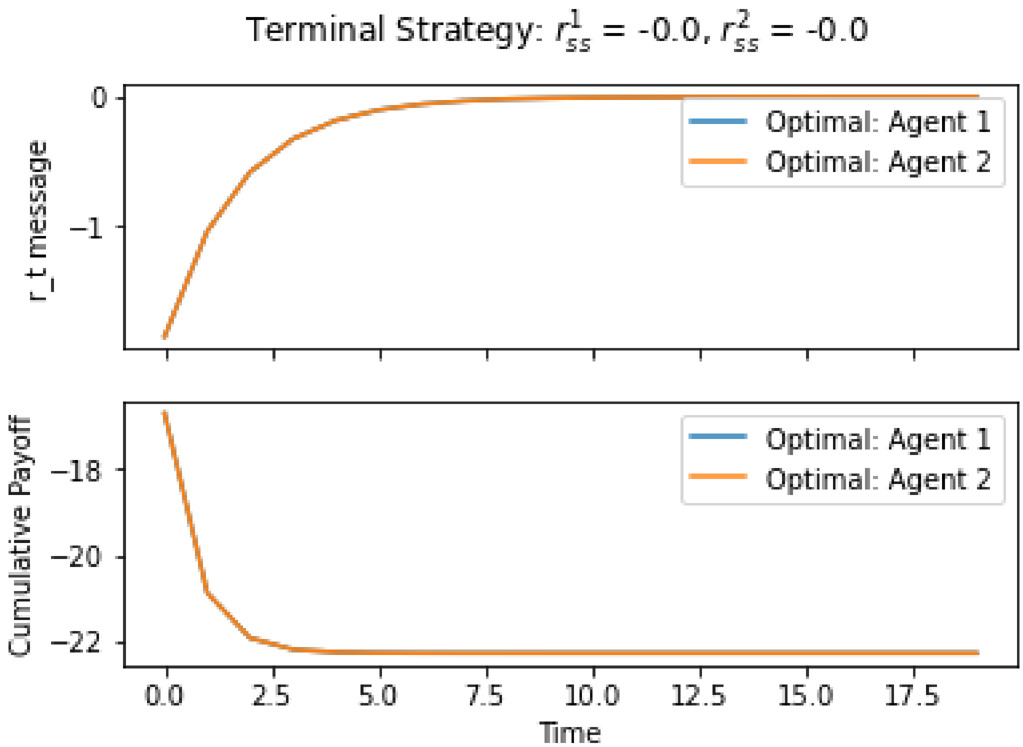
[13]:
```
A = np.array([
   [0.7],
], ndmin = 2)

B_1 = np.array([
   0.15, # split the channel of the strategic agent from the previous model␣
 ↪amongst the two strategic agents here
], ndmin = 2).T

B = [B_1, B_1]

X_0_1 = np.array([
   4,
```

```
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 1
m = 1
L = 2 # two agents now
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 0] # identical agents
r = [0, 0]
c_base = sum([B[l] @ np.array([[r[l]]], ndmin = 2) for l in range(L)])
c = [c_base + (A - np.identity(n)) @ (x[l] * np.ones((n, 1))) for l in range(L)]
```

[14]:
```
max_distances, historical_K, historical_k, historical_kappa = solve(Q, [np.
↪zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x, c,␣
↪tol = 1000)
converge_plot(max_distances, tol = 1000)
```



[15]:
```
xs2, rs2, payoffs2 = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs2, r, payoffs2, set_cap = 20)
```



Clearly as the strategic agents are symmetric, they have symmetric strategies.

[16]:
```
rs[0][:20]
```

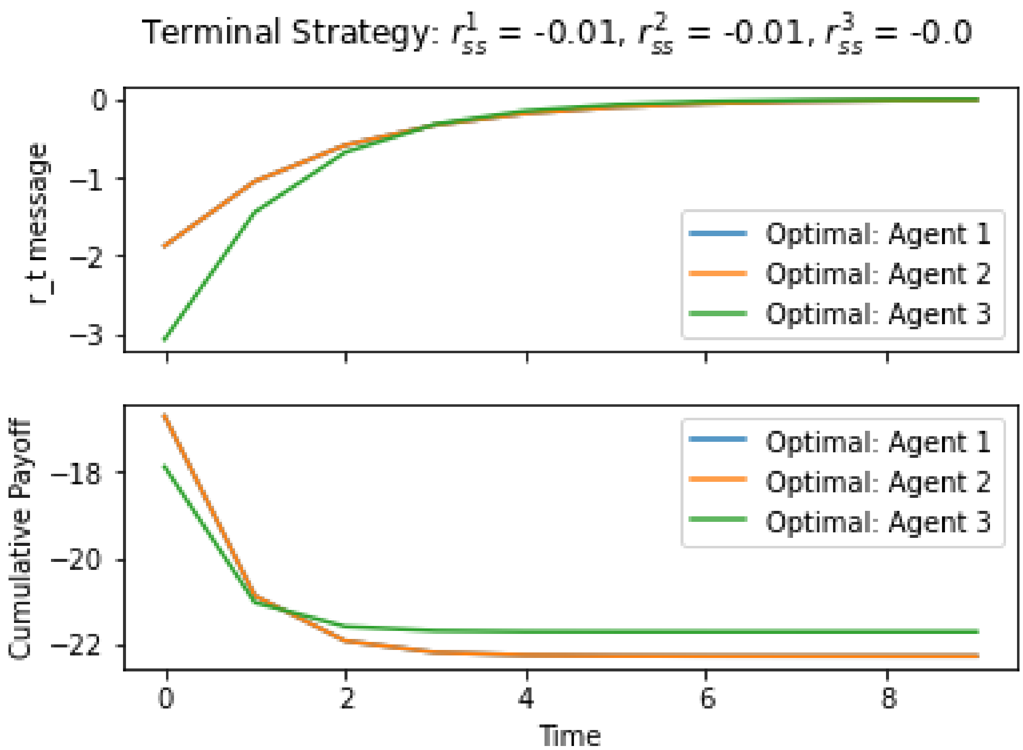
[16]:
```
[array([[-3.06379432]]),
 array([[-1.44064335]]),
 array([[-0.67741273]]),
 array([[-0.31852991]]),
 array([[-0.14977768]]),
 array([[-0.07042778]]),
 array([[-0.03311623]]),
 array([[-0.01557176]]),
 array([[-0.00732208]]),
 array([[-0.00344296]]),
 array([[-0.00161893]]),
 array([[-0.00076125]]),
 array([[-0.00035795]]),
 array([[-0.00016831]]),
 array([[-7.91436038e-05]]),
 array([[-3.72145434e-05]]),
 array([[-1.74988524e-05]]),
 array([[-8.22823031e-06]]),
 array([[-3.86904082e-06]]),
```

```
array([[-1.81928268e-06]])]
```

[17]:
```
rs2[0][:10]
```

[17]:
```
[array([[-1.87084846]]),
 array([[-1.04708837]]),
 array([[-0.58604109]]),
 array([[-0.32799921]]),
 array([[-0.18357668]]),
 array([[-0.10274537]]),
 array([[-0.05750518]]),
 array([[-0.03218487]]),
 array([[-0.01801343]]),
 array([[-0.01008187]])]
```

[18]:
```
L = 3
rs2[2] = rs[0]
payoffs2[2] = payoffs[0]
r_save = [0, 0, 0]
do_plot(rs2, r_save, payoffs2, set_cap = 10)
```



This is both plots overlapping on the same chart. "Agent 3", the green line, is the single strategic agent from the single-agent model.

[19]:
```
payoffs2[2][-1]
```

[19]:
```
-21.719082733041443
```

is the one-agent payoff and:

[20]:
```
payoffs2[0][-1]
```

[20]:
```
-22.28448611624381
```

is the two-agent payoff.