MultipleInfluence3

July 27, 2021

1 Competitive Influence Model: Variations of Cost Weight

James Yu, 27 July 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 Original Setup: one-agent and two-agent models with equal weight values on opinions and messages

Note that under these parameters, the magnitude of the cost of the opinions exceeds that of the messages.

[8]:
```
A = np.array([
  [0.217,    0.2022,    0.2358,    0.1256,    0.1403],
  [0.2497,    0.0107,    0.2334,    0.1282,    0.378],
  [0.1285,    0.0907,    0.3185,    0.2507,    0.2116],
  [0.1975,    0.0629,    0.2863,    0.2396,    0.2137],
```

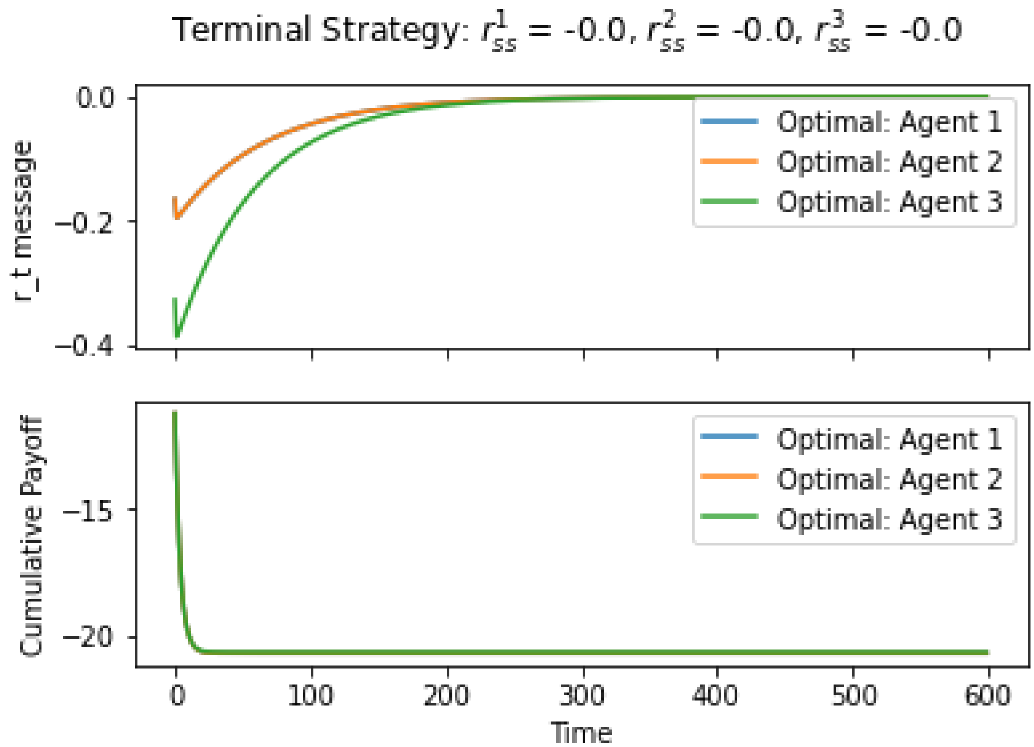
```
  [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
], ndmin = 2)

B_1 = np.array([
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1]

X_0_1 = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1]

delta = 0.8
n = 5
m = 1
L = 1
Q = [0.2 * np.identity(n)]
R = [0.2 * np.identity(m)]

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)]
```

[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, tol = 1000)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
save_rs = rs
save_payoffs = payoffs
save_K = historical_K
save_k = historical_k
save_kappa = historical_kappa
save_xs = xs
```

[10]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   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)

B_1 = np.array([
  0.03955, # split /2, let B_1 = B_2
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_1]

X_0_1 = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 5
m = 1
L = 2 # two agents now
Q = [0.2 * np.identity(n), 0.2 * 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)]
```

[11]:
```
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)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
L = 3
rs[2] = save_rs[0]
payoffs[2] = save_payoffs[0]
r_save = [0, 0, 0]
do_plot(rs, r_save, payoffs, set_cap = 600)
```



Recall that **agent 3** is the **agent 1 from the one-agent model**, as used in the previous notebook.

1.2 New Setup: one-agent and two-agent models with smaller message cost (R = 0.01)

[12]:
```
A = np.array([
  [0.217,     0.2022,    0.2358,    0.1256,    0.1403],
  [0.2497,    0.0107,    0.2334,    0.1282,    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)

B_1 = np.array([
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1]

X_0_1 = np.array([
```

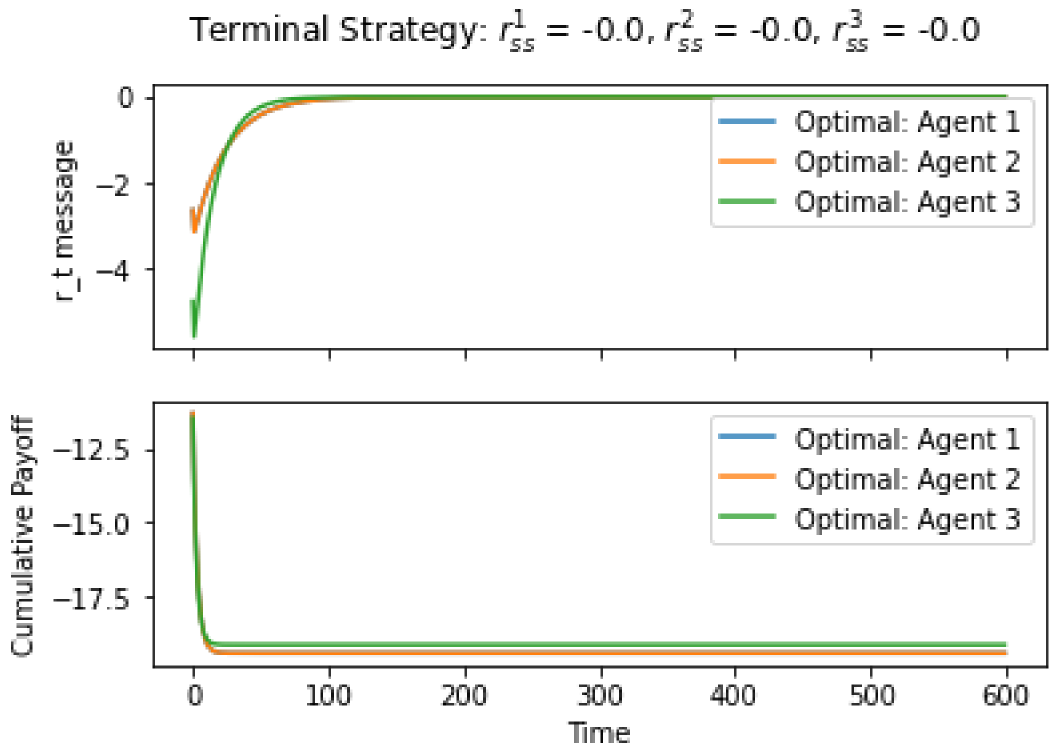
```
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1]

delta = 0.8
n = 5
m = 1
L = 1
Q = [0.2 * np.identity(n)]
R = [0.01 * np.identity(m)] # new value

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)]
```

[13]:
```
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, tol = 1000)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
save_rs = rs
save_payoffs = payoffs
save_K = historical_K
save_k = historical_k
save_kappa = historical_kappa
save_xs = xs
```

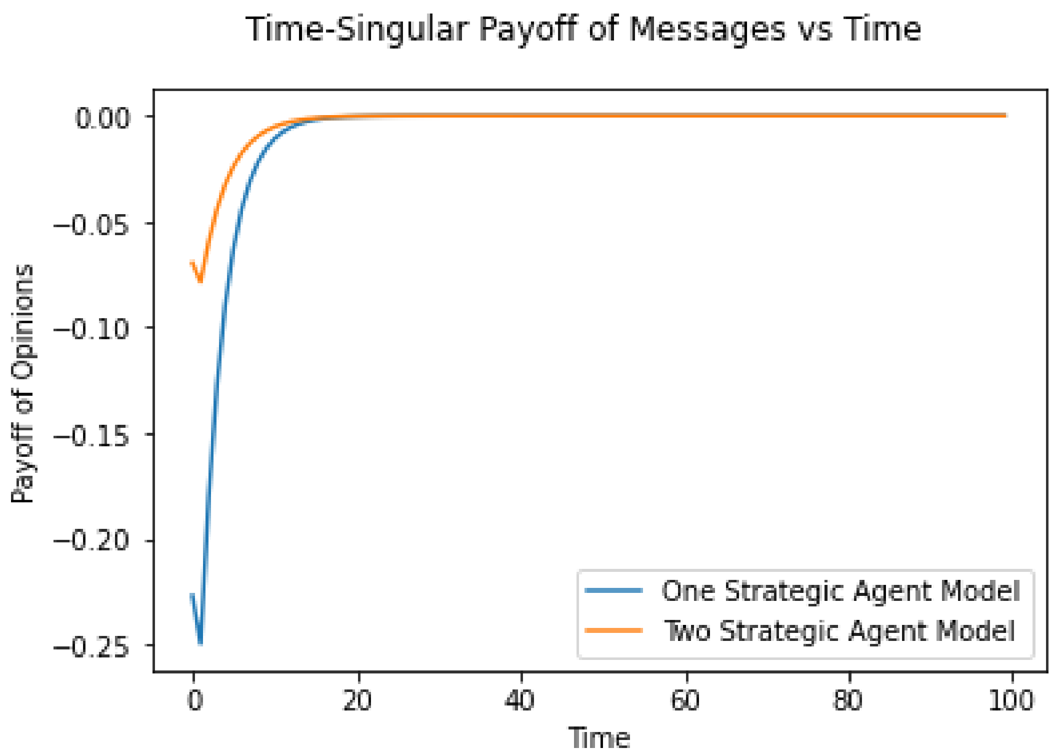
[14]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   0.378],
  [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
  [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
  [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
], ndmin = 2)

B_1 = np.array([
  0.03955, # split /2, let B_1 = B_2
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_1]
```

```
X_0_1 = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 5
m = 1
L = 2 # two agents now
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.01 * np.identity(m), 0.01 * np.identity(m)] # changed this

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)]
```

[15]:
```
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)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
L = 3
rs[2] = save_rs[0]
payoffs[2] = save_payoffs[0]
r_save = [0, 0, 0]
do_plot(rs, r_save, payoffs, set_cap = 600)
```



[16]:
```
fig, ax = plt.subplots()
fig.suptitle("Time-Singular Payoff of Messages vs Time")
plt.plot(range(100), [-delta**i  * (r_i.T @ R[0] @ r_i).item() for i, r_i in␣
 ↪enumerate(save_rs[0][:100])], label = "One Strategic Agent Model")
plt.plot(range(100), [-delta**i  * (r_i.T @ R[0] @ r_i).item() for i, r_i in␣
 ↪enumerate(rs[0][:100])], label = "Two Strategic Agent Model")
plt.xlabel("Time")
plt.ylabel("Payoff of Opinions")
ax.legend()
plt.show()
```



1.3 Another new setup: higher message cost (R = 5)

[17]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   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)

B_1 = np.array([
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1]

X_0_1 = np.array([
  -0.98,
```

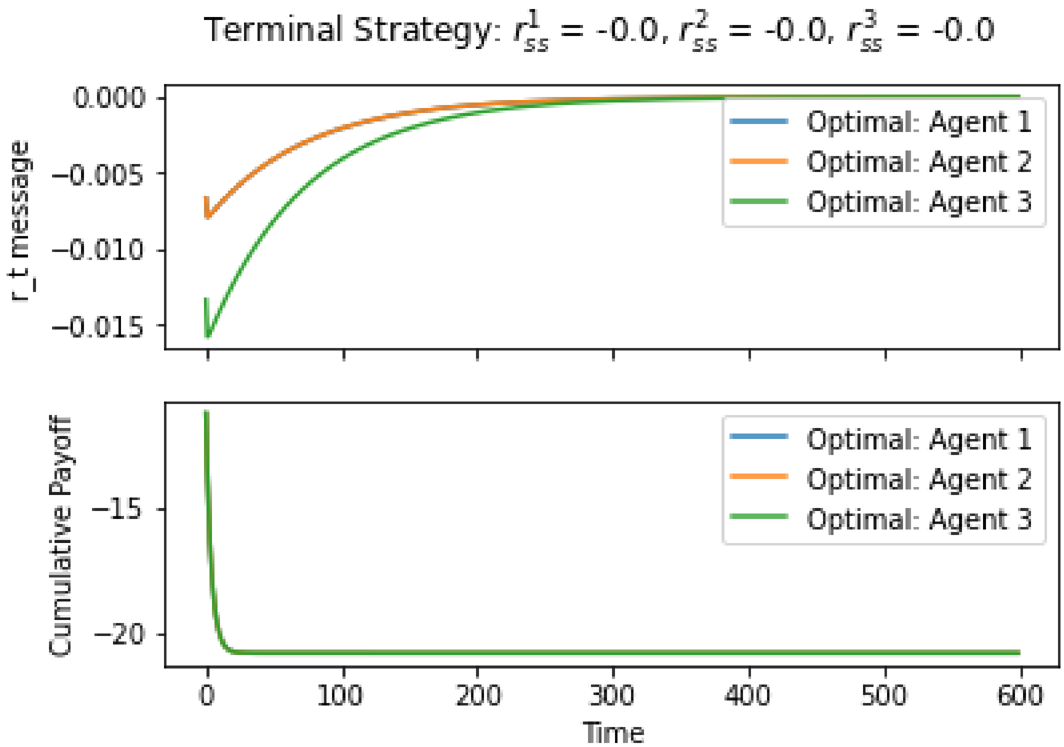
```
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1]

delta = 0.8
n = 5
m = 1
L = 1
Q = [0.2 * np.identity(n)]
R = [5 * np.identity(m)] # new value

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)]
```

[18]:
```
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, tol = 1000)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
save_rs = rs
save_payoffs = payoffs
save_K = historical_K
save_k = historical_k
save_kappa = historical_kappa
save_xs = xs
```

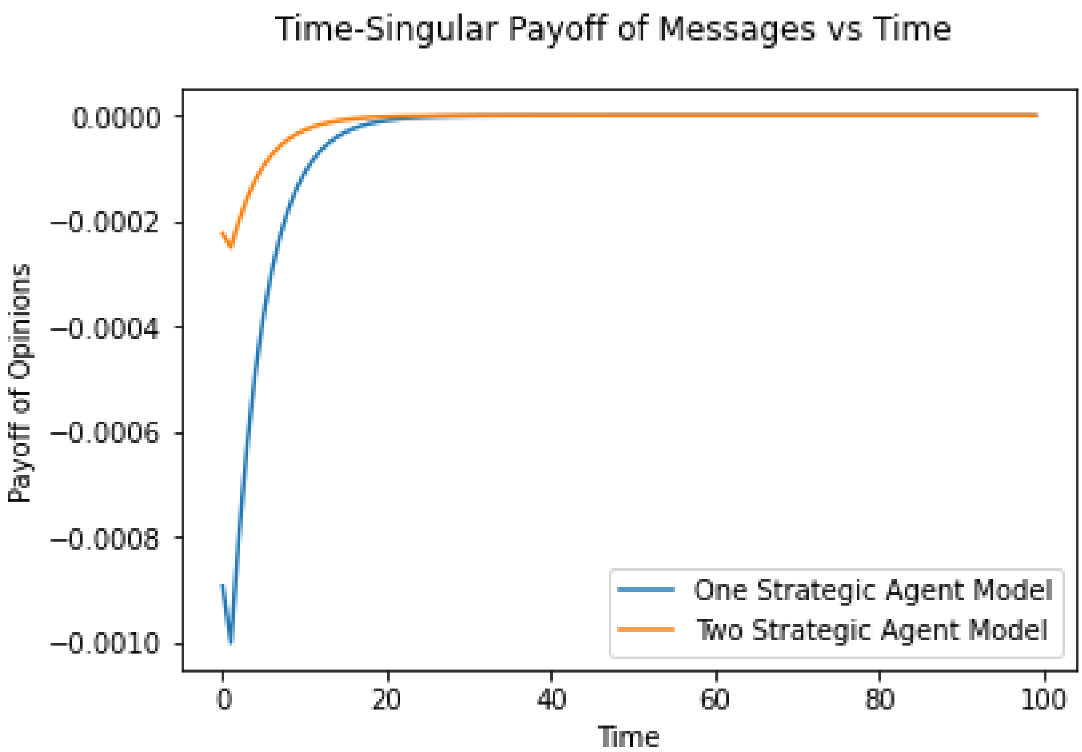
[19]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   0.378],
  [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
  [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
  [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
], ndmin = 2)

B_1 = np.array([
  0.03955, # split /2, let B_1 = B_2
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_1]

X_0_1 = np.array([
```

```
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 5
m = 1
L = 2 # two agents now
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [5 * np.identity(m), 5 * np.identity(m)] # changed this

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)]
```

[20]:
```
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)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
L = 3
rs[2] = save_rs[0]
payoffs[2] = save_payoffs[0]
r_save = [0, 0, 0]
do_plot(rs, r_save, payoffs, set_cap = 600)
```



[21]:
```
fig, ax = plt.subplots()
fig.suptitle("Time-Singular Payoff of Messages vs Time")
plt.plot(range(100), [-delta**i  * (r_i.T @ R[0] @ r_i).item() for i, r_i in␣
 ↪enumerate(save_rs[0][:100])], label = "One Strategic Agent Model")
plt.plot(range(100), [-delta**i  * (r_i.T @ R[0] @ r_i).item() for i, r_i in␣
 ↪enumerate(rs[0][:100])], label = "Two Strategic Agent Model")
plt.xlabel("Time")
plt.ylabel("Payoff of Opinions")
ax.legend()
plt.show()
```



1.4 Conclusion:

- Higher cost values on the messages result in less costly message magnitudes being used, which offsets the effect.
- Lower cost values on the messages result in more costly message magnitudes being used (because even though they are slightly more expensive in the short-run, they have no cost in the long-run due to discounting).

1.5 Extra Test: Limit as cost of messages goes to zero:

[22]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   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)

B_1 = np.array([
  0.0791,
  0,
  0,
```

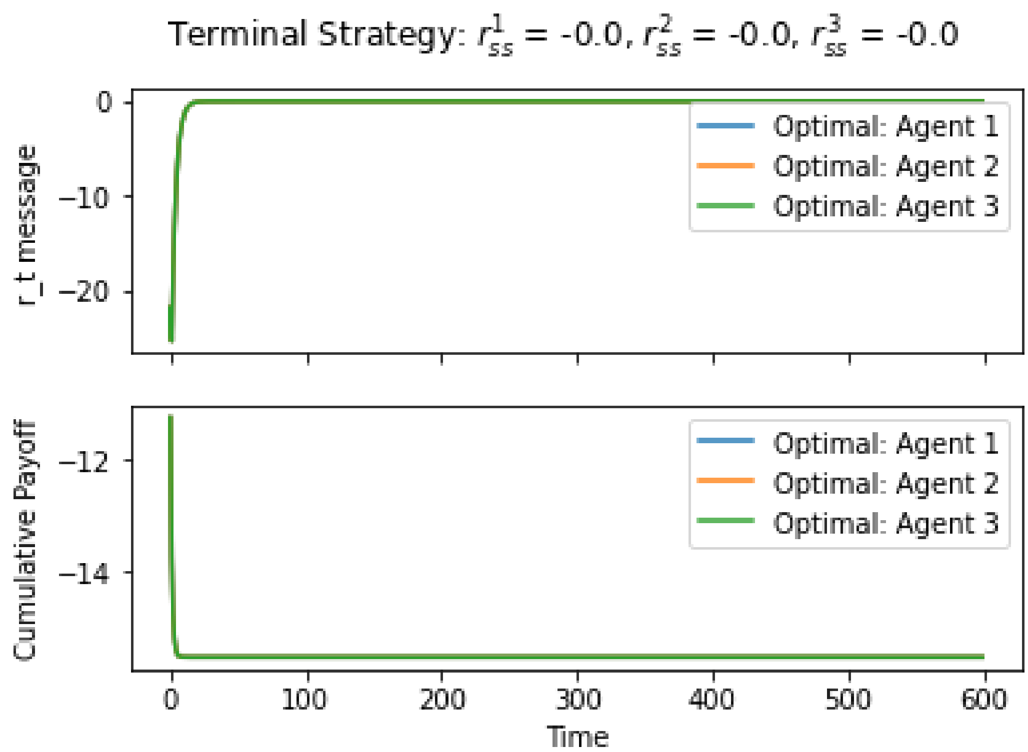
```
  0,
  0,
], ndmin = 2).T

B = [B_1]

X_0_1 = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1]

delta = 0.8
n = 5
m = 1
L = 1
Q = [0.2 * np.identity(n)]
R = [0.00000000001 * np.identity(m)] # new value

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)]
```

[23]:
```
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, tol = 1000)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
save_rs = rs
save_payoffs = payoffs
save_K = historical_K
save_k = historical_k
save_kappa = historical_kappa
save_xs = xs
```

[24]:
```
A = np.array([
   [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
   [0.2497,   0.0107,   0.2334,   0.1282,   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)

B_1 = np.array([
  0.03955, # split /2, let B_1 = B_2
  0,
```

```
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_1]

X_0_1 = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 5
m = 1
L = 2 # two agents now
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.00000000001 * np.identity(m), 0.00000000001 * np.identity(m)] # changed␣
↪this

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)]
```

[25]:
```
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)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
L = 3
rs[2] = save_rs[0]
payoffs[2] = save_payoffs[0]
r_save = [0, 0, 0]
do_plot(rs, r_save, payoffs, set_cap = 600)
```

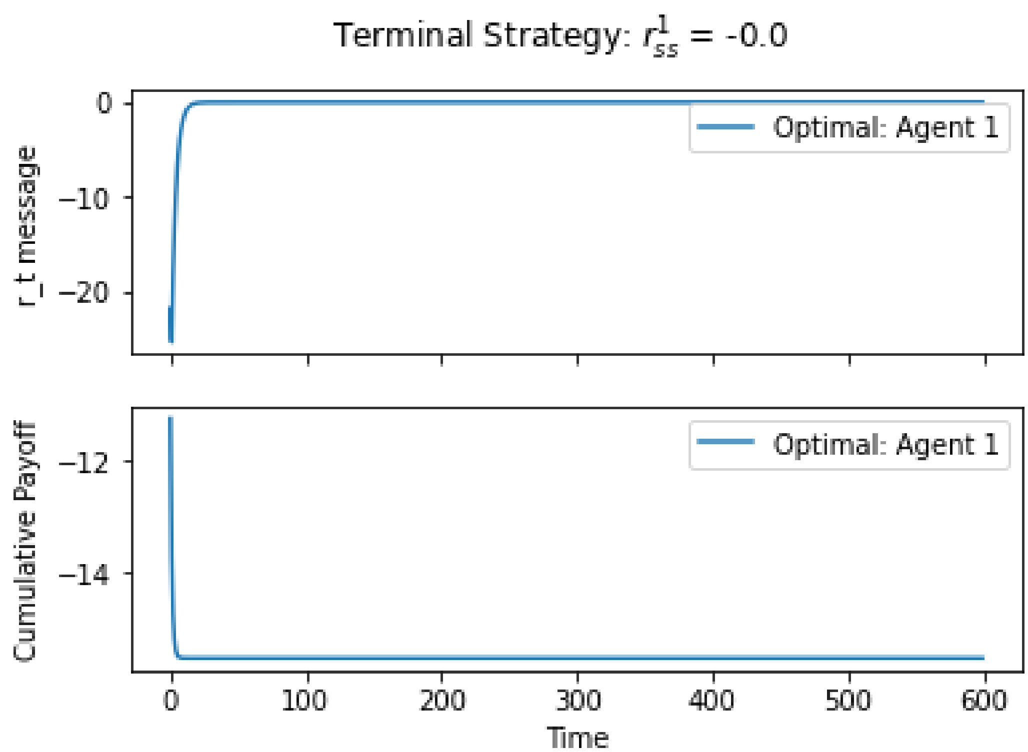


1.6 BENCHMARK: One Agent, No cost at all:

[27]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   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)

B_1 = np.array([
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1]

X_0_1 = np.array([
  -0.98,
```

```
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1]

delta = 0.8
n = 5
m = 1
L = 1
Q = [0.2 * np.identity(n)]
R = [0 * np.identity(m)] # NO 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)]
```

[28]:
```
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, tol = 1000)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, set_cap = 600)
```



1.7 Conclusion:

In the limit, the Nash equilibrium is the one-agent strategy.

1.8 Test Of the Other Side: Limit as cost goes to infinity

[29]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   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)

B_1 = np.array([
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1]

X_0_1 = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1]

delta = 0.8
n = 5
m = 1
L = 1
Q = [0.2 * np.identity(n)]
R = [1000 * np.identity(m)] # new value

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)]
```

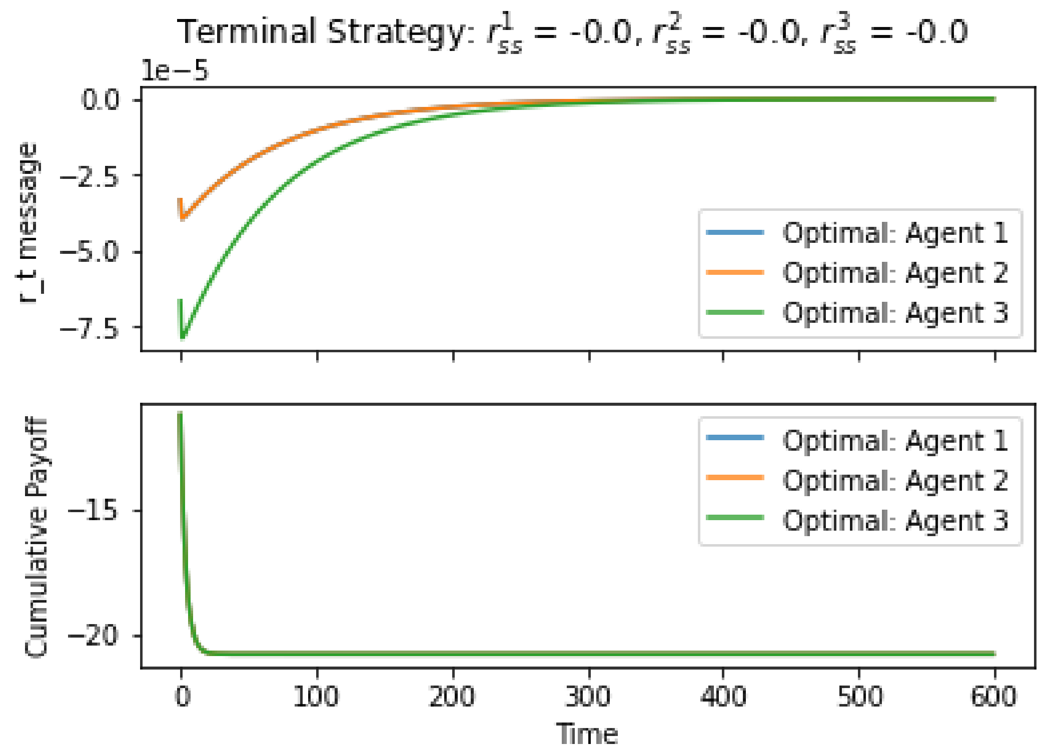
[30]:
```
max_distances, historical_K, historical_k, historical_kappa = solve([Q[0]], [np.
 ↪zeros((1, n))], [0], A, B, delta, n, m, L, Q, R, x, c, tol = 1000)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
```

```
save_rs = rs
save_payoffs = payoffs
save_K = historical_K
save_k = historical_k
save_kappa = historical_kappa
save_xs = xs
```

[31]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   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)

B_1 = np.array([
  0.03955, # split /2, let B_1 = B_2
  0,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_1]

X_0_1 = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 5
m = 1
L = 2 # two agents now
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [1000 * np.identity(m), 1000 * np.identity(m)] # changed this

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)]
```

[32]:

```
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)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
L = 3
rs[2] = save_rs[0]
payoffs[2] = save_payoffs[0]
r_save = [0, 0, 0]
do_plot(rs, r_save, payoffs, set_cap = 600)
```



1.9 NOTE: The 1e-5 means 10ˆ-5. That is, the axis scaling for the upper chart is -5 * 10ˆ-5, -2.5 * 10ˆ-5, etc. The lower chart is the same.

1.10 Conclusion

As cost becomes prohibitively high, the trajectories still have this half-size split, but the messages go to zero (which makes sense) which drives the payoffs to being identical to the one-agent case at infinity.