# MultipleInfluence2

July 27, 2021

## 1 Additional Tests of the Competitive Influence Model

James Yu, 26 July 2021

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

[2]:
```python
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_
```

```
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]:
```
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]:
```
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()
```

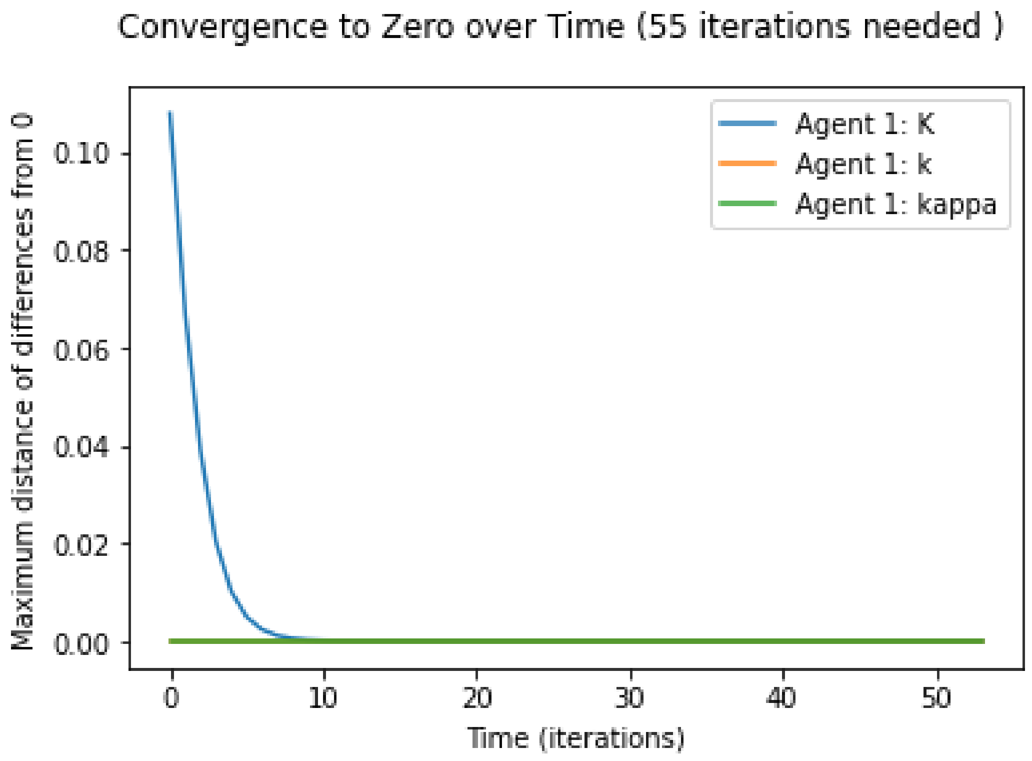
## 1.1 Modification of Test 1 (simple 5-naive 1-strategic network without bot):

**Given that $K_t$ converges, it must be the case that $r_{ss}$ under the optimal strategy of $K_{ss}$ is optimal**

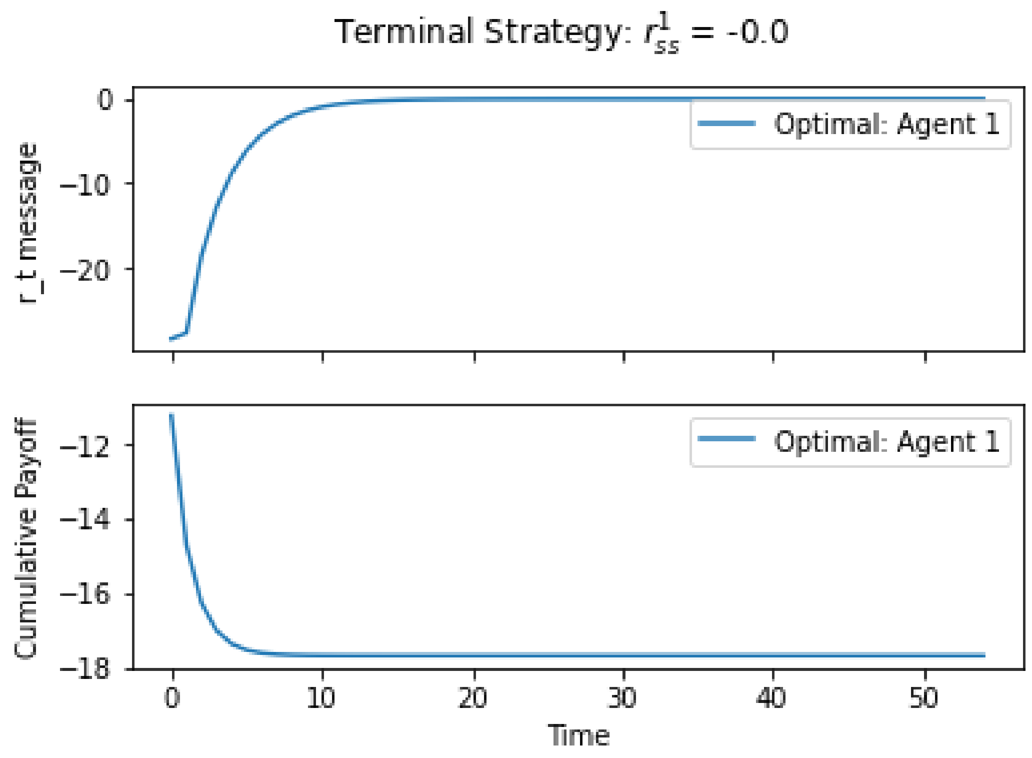
[8]:
```
A = np.array([
   [0.217,     0.2022,    0.2358,    0.1256,    0.1403],
   [0.2497,    0.0107,    0.2334,    0.1282,    0.378],
   [0.1285,    0.0907,    0.3185,    0.2507,    0.2116],
   [0.1975,    0.0629,    0.2863,    0.2396,    0.2137],
   [0.1256,    0.0711,    0.0253,    0.2244,    0.5536],
], ndmin = 2)
```

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

B = [B_1]

delta = 1
n = 5
m = 1
L = 1
Q = [0.2 * np.identity(n)]
R = [0 * np.identity(m)] # NO COST (like the original model)

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([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T # here \chi_0 = initial opinions as agenda x_0 = 0
X_0 = [X_0_1]
```

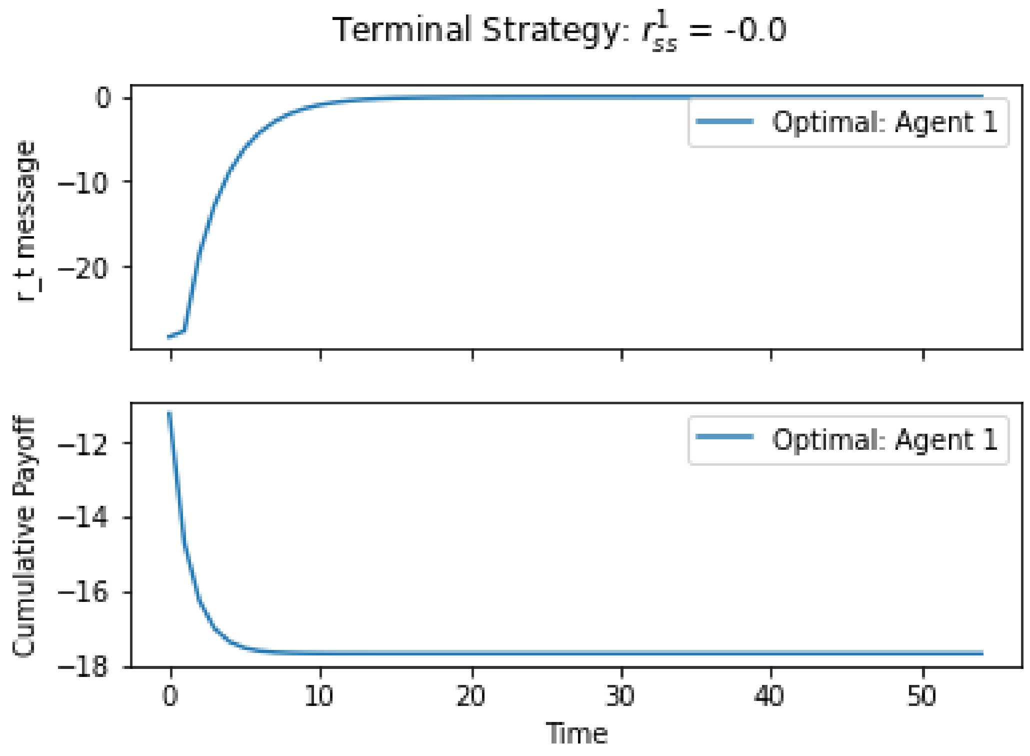
[9]:
```
max_distances, historical_K, historical_k, historical_kappa = solve([Q[0]], [np.
 ↪zeros((1, n))], [0], A, B, delta, n, m, L, Q, R, x, c)
converge_plot(max_distances)
```



[10]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, set_cap = 55)
```



[11]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa,␣
 ↪infinite = False)
do_plot(rs, r, payoffs)
```



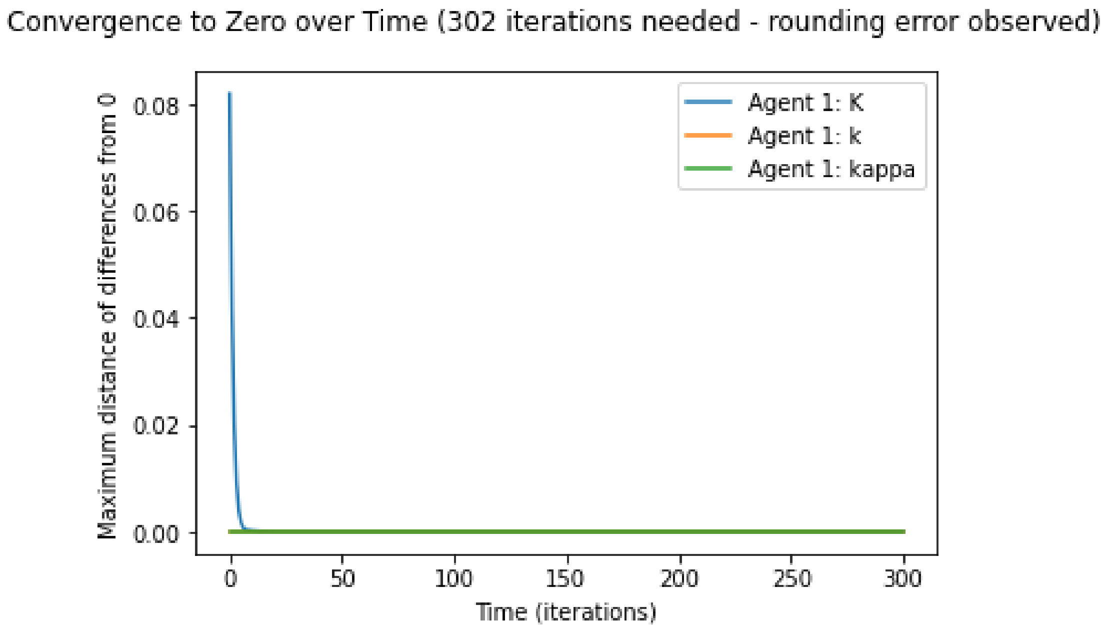
They are the same plot, so the $r_{ss}$ method works.

## 1.2 Modification of Test 2 (simple 5-naive 1-strategic network with bot)

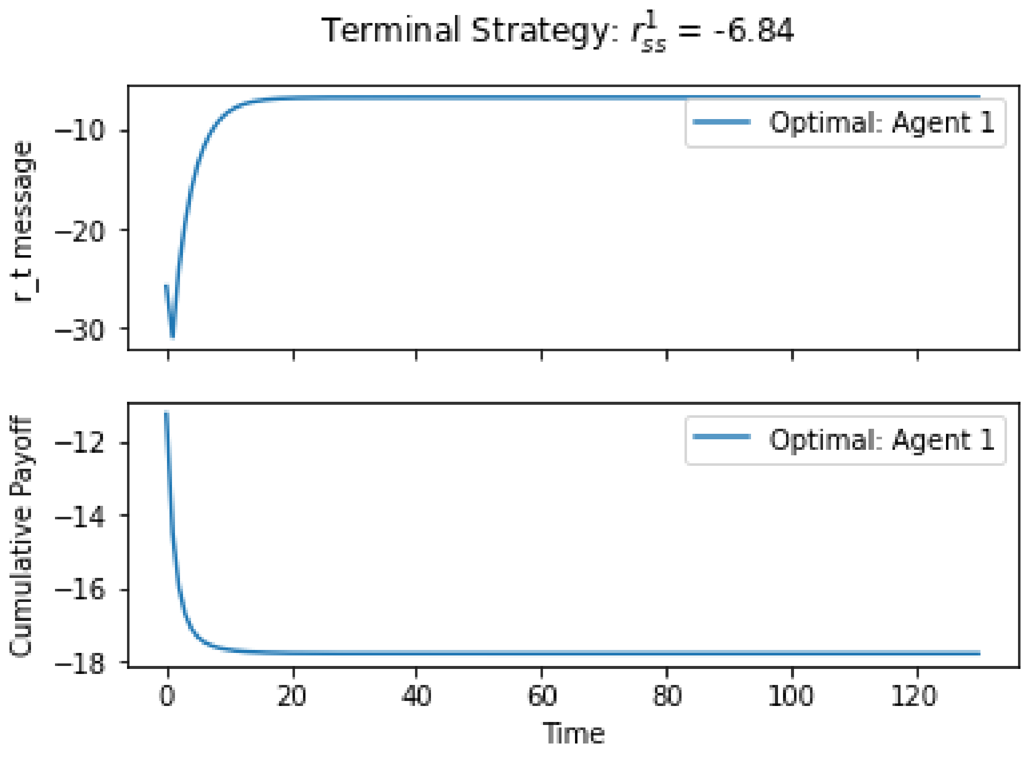
[12]:
```
A = np.array([
  [1,        0,        0,        0,        0,        0     ], # the bot
  [0,        0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.1012,   0.8988*0.2497,   0.8988*0.0107,   0.8988*0.2334,   0.8988*0.1282, ␣
 ↪ 0.8988*0.378 ],
  [0,        0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
  [0,        0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
  [0,        0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
], ndmin = 2)

B_1 = np.array([
  0, # the bot
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T
```

```
B = [B_1]

delta = 0.8 # delta set here
n = 6       # 6 technical naive agents (5 + bot)
m = 1
L = 1
Q = [0.2 * np.identity(n)]
Q[0][0, :] = 0 # ADJUST FOR BOT
R = [0 * np.identity(m)] # STILL 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)]

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

[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)
converge_plot(max_distances)
```

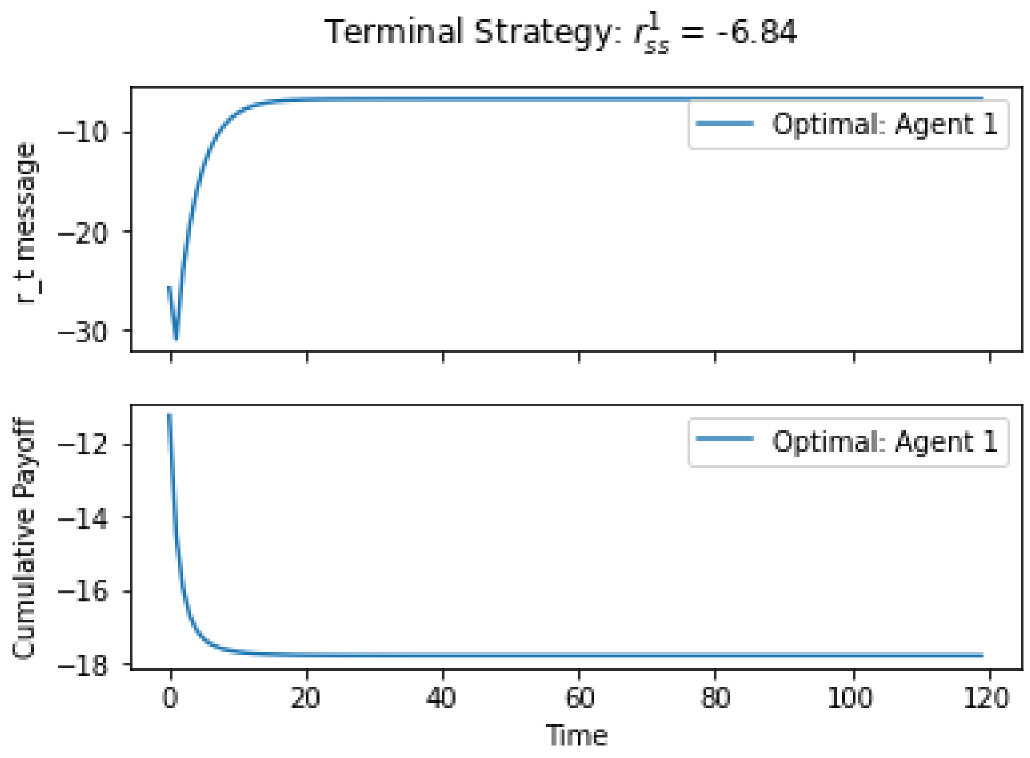


[14]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs)
```



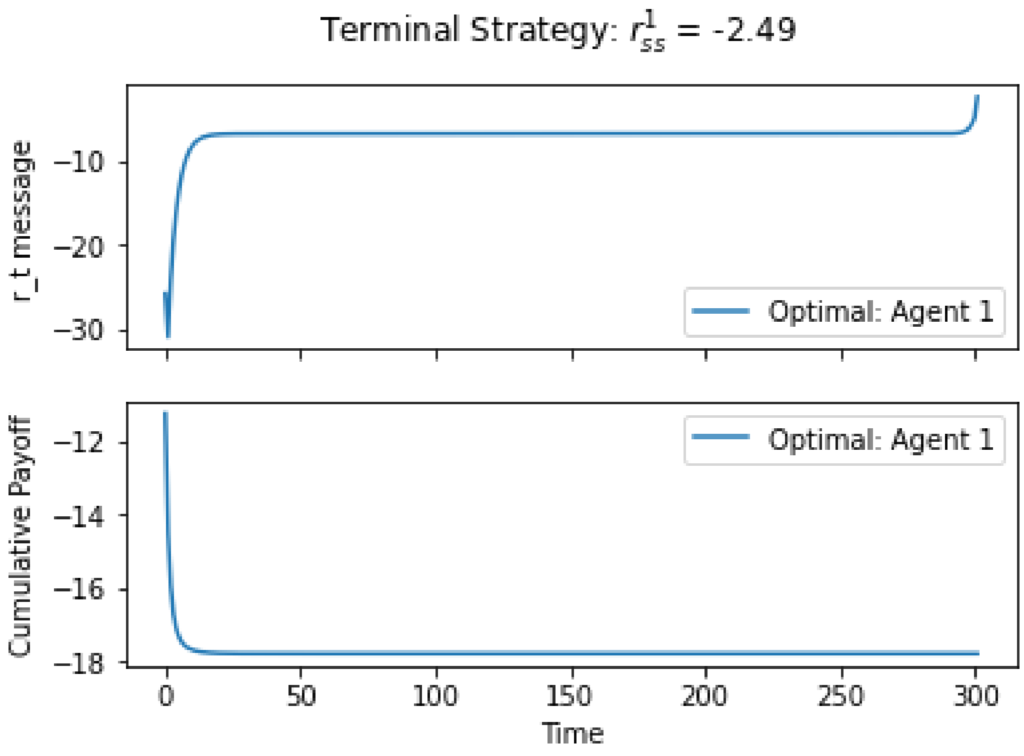
Compare to before:

[15]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa,␣
 ↪infinite = False)
do_plot(rs, r, payoffs, set_cap = 120)
```



They are the same plot. But now, if I let the original plot run through every $K_t$ recorded:

[16]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa,␣
 ↪infinite = False)
do_plot(rs, r, payoffs)
```



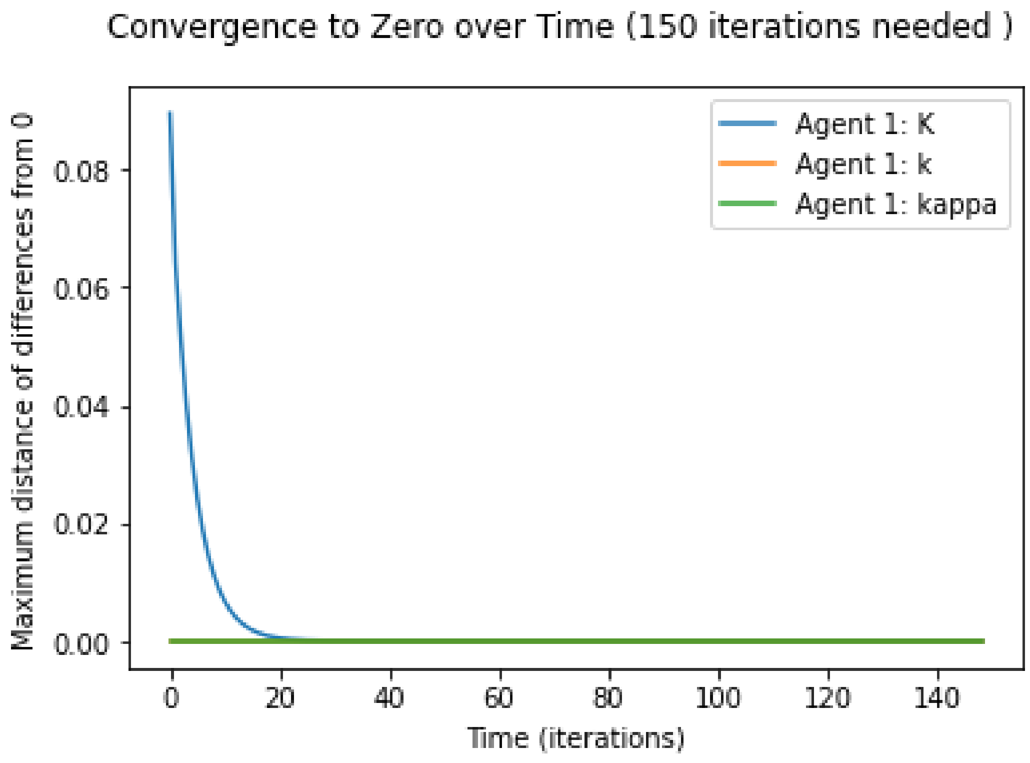
The finding is that because convergence of the opinions does not happen at the same speed as convergence of $r_t$, we have in the original case a finite-horizon solution set to however many periods it took for the messages to converge. This does not accurately reflect the long-run however as in the long run, unlike with using a finite number of time intervals, the network doesn't suddenly stop broadcasting (people communicate forever), so this tail cannot be optimal in that sense because then the bot/etc would be at an advantage with future time intervals where the strategic agent stops broadcasting.

## 1.3 Modified Test 4: Single Strategic Agent, 5-agent Setup, Induce Cost (now infinite and with delta = 0.8)

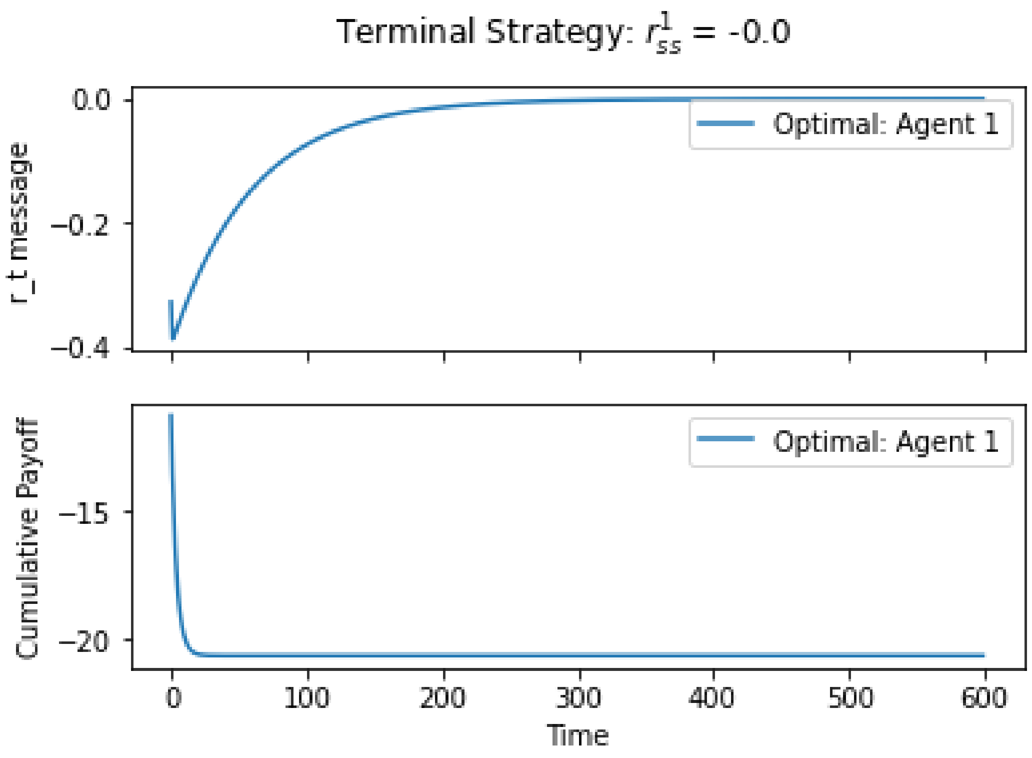
**Much of the work in this section (and test 5) is a bit messy; there is a clean summary at the end of Test 5.**

[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 = [0.2 * np.identity(m)] # NOW WITH 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)]
```

[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)
converge_plot(max_distances, tol = 1000)
```



[19]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, set_cap = 600)
```

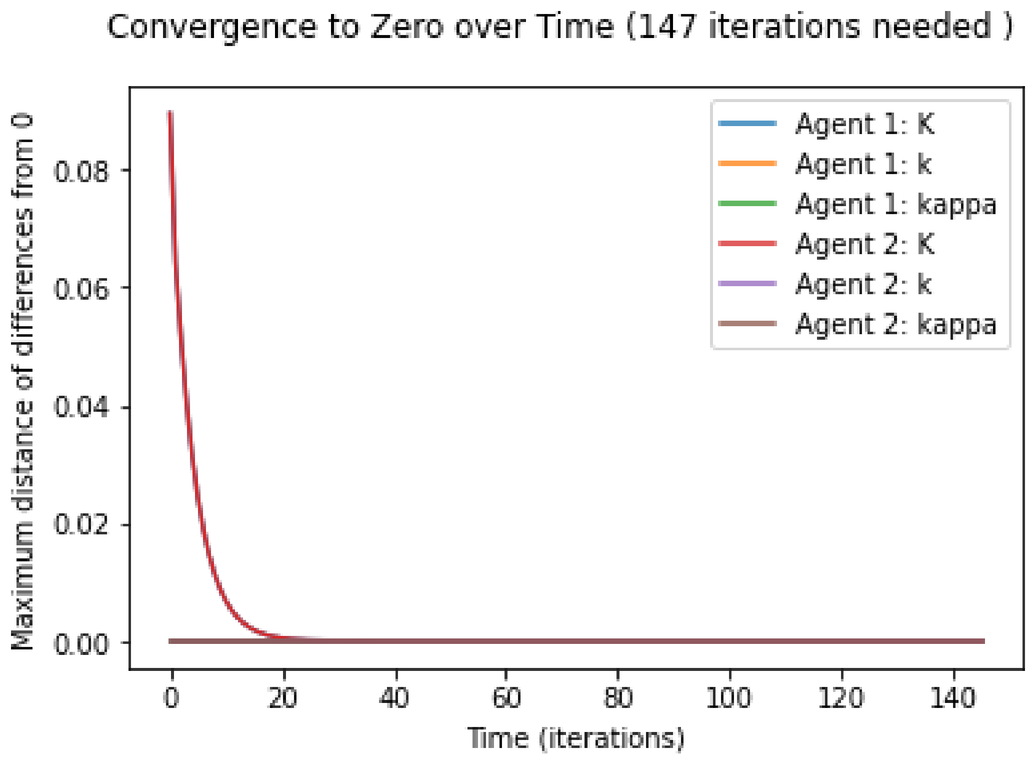


[20]:
```
save_rs = rs
save_payoffs = payoffs
save_K = historical_K
save_k = historical_k
save_kappa = historical_kappa
save_xs = xs
```

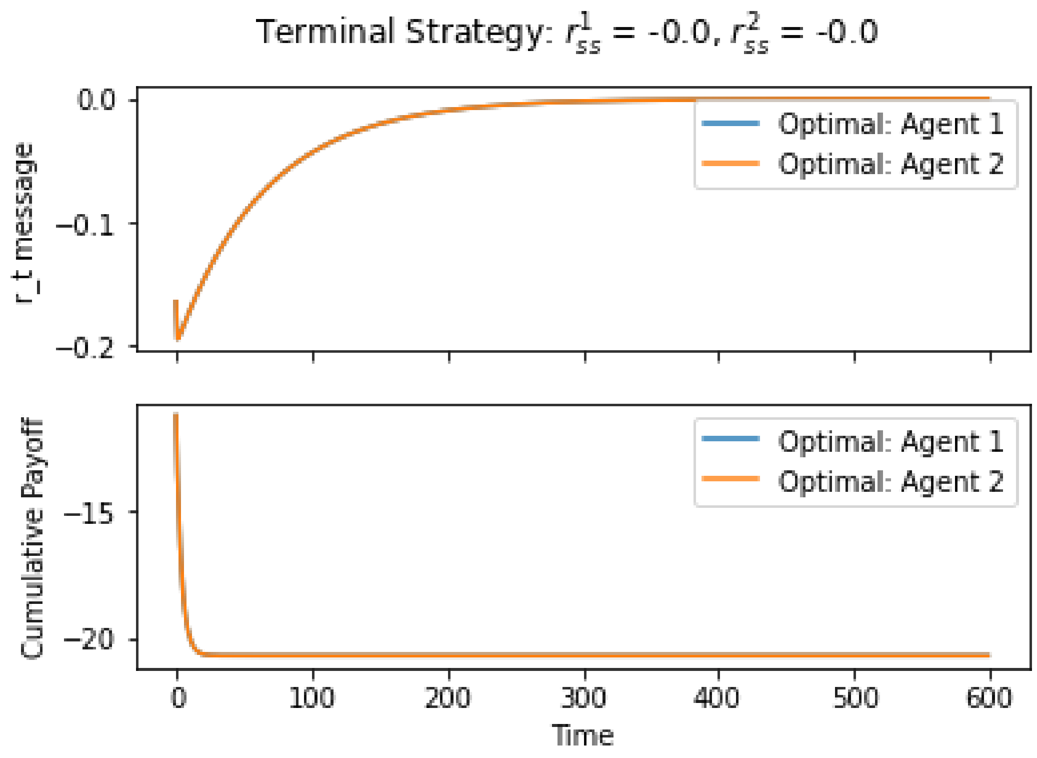
### 1.4 Modified Test 5: Dual Strategic Agent, Equal Agendas, Split Influence to Agent 1 (now infinite and with delta = 0.8)

[21]:
```
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)]
```

[22]:
```
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)
```



[23]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, set_cap = 600)
```

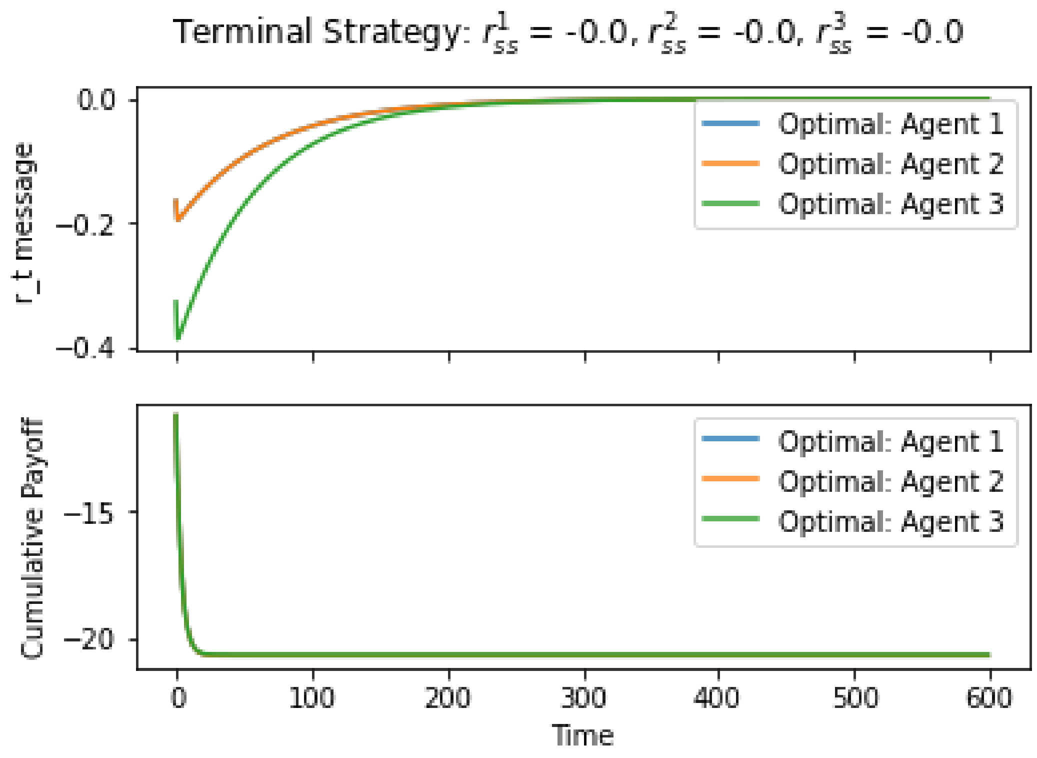


[24]:
```
xs[0][-1]
```

[24]:
```
array([[1.83e-322],
       [1.93e-322],
       [2.03e-322],
       [1.98e-322],
       [2.03e-322]])
```

This becomes more readable if I plot the original and new graphs on the same plot:

[25]:
```
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)
```



Here "agent 3" is the one from the single-agent variant of the problem. Notice how in the single-agent variant, $r_t$ is generally further away from the agenda of 0. This should fall under intuition as when two agents have the same agenda, they can achieve the same goal by splitting work evenly. Notice the orange line gets close to zero slightly faster than the green line; they do converge at around the same rates however.

To analyze this more specifically, observe the payoffs for each agent:

[26]:
```
max([abs(i - j) for i, j in zip(payoffs[0], payoffs[1])])
```

[26]:
```
0.0
```

This says that when comparing the payoffs for the two strategic agents (0 and 1) in the two-agent problem, the maximum difference at any time point between their cumulative payoff is zero. This means indeed the two agents are identical in that regard. However, when comparing one of them to the strategic agent in the one-agent problem, we get:

[27]:
```
max([abs(i - j) for i, j in zip(payoffs[0], payoffs[2])])
```

[27]:
```
0.036456742706084455
```

This has some difference. Specifically:

[28]:
```
",".join([str(a) for a in [i - j for i, j in zip(payoffs[0], payoffs[2])][:
↪150]])
```

[28]:
```
'0.015979736200678 83,0.032328590686830694,0.036456742706084455,0.035236842192656
326,0.03102559729291343,0.025269983784603767,0.01888211895179026,0.0124394623891
53277,0.0062910205395319,0.0006312655682201296,-0.004447904830062299,-0.00892081
```

```
1992947364,-0.012802820156583294,-0.016133145124094028,-0.018963326425204485,-0.
021349683413149023,-0.023348545190710723,-0.02501338736792036,-0.026393258540096
554,-0.02753206066275027,-0.028468378420253515,-0.02923564673371004,-0.029862512
6995411,-0.030373295842913706,-0.030788484011534223,-0.031125225486725583,-0.031
39779384407149,-0.031618012862796974,-0.03179563588948042,-0.03193867861059019,-
0.03205370697138932,-0.03214608356087112,-0.032220176563299674,-0.03227953563749
9635,-0.03232703901999301,-0.03236501589103824,-0.0323953476832628,-0.0324195516
10168185,-0.03243884928387786,-0.03245422290111932,-0.032466461116190004,-0.0324
76196396025614,-0.0324839353670896,-0.032490083415822824,-0.03249496459164547,-0
.032498837680520865,-0.0325019091643064,-0.03250434365333632,-0.0325062722729718
7,-0.03250779939650883,-0.032509008043916765,-0.03250996420577934,-0.03251072030
2752816,-0.03251131795058271,-0.03251179016798389,-0.03251216313810801,-0.032512
457612661194,-0.03251269003029478,-0.03251287340671993,-0.03251301804271023,-0.0
32513132086766205,-0.03251322198208939,-0.03251329282137405,-0.03251334862831356
5,-0.03251339258077479,-0.032513427187705446,-0.03251345442923537,-0.03251347586
761,-0.032513492734992155,-0.03251350600288916,-0.03251351643706357,-0.032513524
640933156,-0.032513531089843895,-0.032513536158166545,-0.03251354014065555,-0.03
251354326932798,-0.032513545726768456,-0.03251354765662384,-0.032513549171891754
,-0.03251355036142556,-0.03251355129508937,-0.03251355202780104,-0.0325135526027
1538,-0.03251355305374659,-0.032513553407532925,-0.03251355368499986,-0.03251355
390257871,-0.03251355407317291,-0.03251355420690771,-0.03251355431173408,-0.0325
13554393887034,-0.03251355445826576,-0.03251355450870719,-0.032513554548227575,-
0.03251355457918237,-0.03251355460342964,-0.032513554622418894,-0.03251355463729
0554,-0.032513554648932796,-0.03251355465804551,-0.032513554665179356,-0.0325135
5467076422,-0.03251355467513406,-0.03251355467855532,-0.032513554681230517,-0.03
2513554683323065,-0.032513554684960866,-0.032513554686243396,-0.0325135546872452
6,-0.03251355468802686,-0.032513554688637925,-0.03251355468911754,-0.03251355468
9490576,-0.0325135546897819,-0.032513554690012825,-0.03251355469019401,-0.032513
55469033257,-0.0325135546904427,-0.03251355469052797,-0.03251355469059547,-0.032
51355469064876,-0.03251355469069139,-0.03251355469072337,-0.032513554690748236,-
0.03251355469076955,-0.032513554690783764,-0.03251355469079442,-0.03251355469080
508,-0.032513554690812185,-0.03251355469081929,-0.032513554690826396,-0.03251355
469082995,-0.0325135546908335,-0.032513554690837054,-0.032513554690837054,-0.032
513554690837054,-0.032513554690837054,-0.03251355469084061,-0.03251355469084061,
-0.03251355469084061,-0.03251355469084061,-0.03251355469084061,-0.03251355469084
061,-0.03251355469084061,-0.03251355469084061,-0.03251355469084061,-0.0325135546
9084061,-0.03251355469084061,-0.03251355469084061,-0.03251355469084061'
```

At first, the cumulative payoff of the agents in the two-agent problem is higher. In the long run, however, the one-agent problem has a higher payoff than the two-agent problem by about 0.03.

Also, this is the per-agent payoff, not the total system payoff, which means in the two-agent problem, the agent still has the same cost.

Recall that we can compute the total cumulative payoff by $-x_0' K_0 x_0 - k_0' x_0 + \kappa_0$, which for the two problems is as follows:

### 1.4.1 Payoff in the One-Agent Model:

```
[29]: (- X_0_1.T @ save_K[0][0] @ X_0_1 - save_k[0][0] @ X_0_1 + save_kappa[0][0]).
      ↪item()
```

```
[29]: -20.666612675948883
```

### 1.4.2 Payoffs in the Two-Agent Model:

```
[30]: (- X_0_1.T @ historical_K[0][0] @ X_0_1 - historical_k[0][0] @ X_0_1 +␣
      ↪historical_kappa[0][0]).item()
```

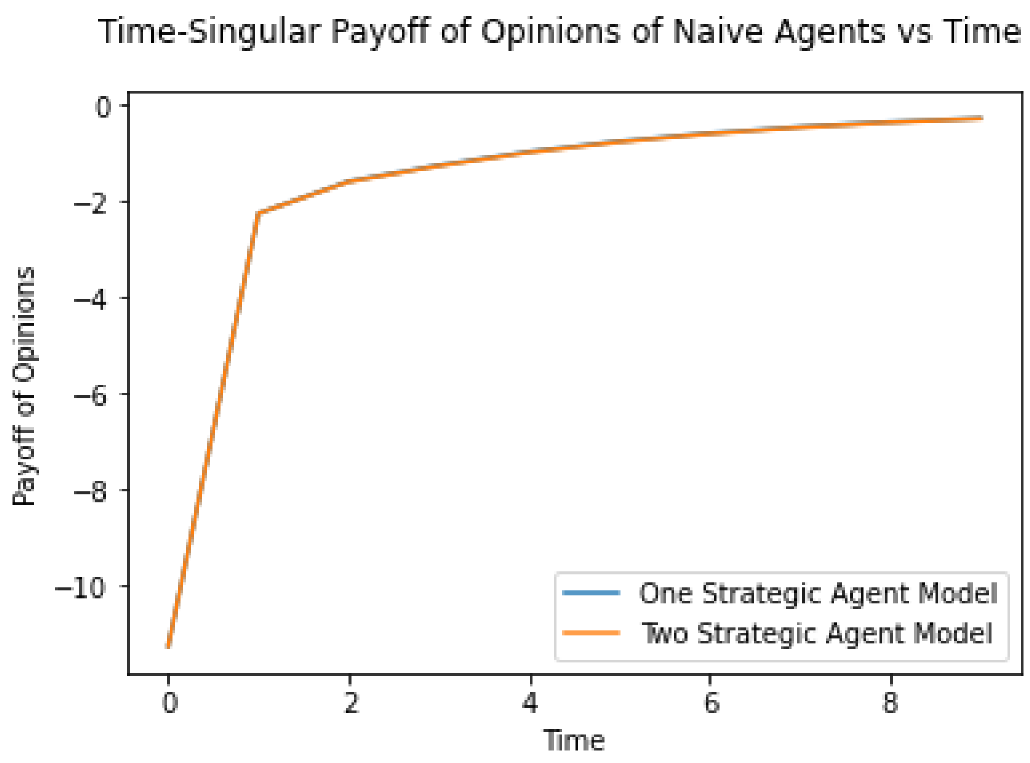
```
[30]: -20.69912623063974
```

```
[31]: (- X_0_1.T @ historical_K[0][1] @ X_0_1 - historical_k[0][1] @ X_0_1 +␣
      ↪historical_kappa[0][1]).item()
```

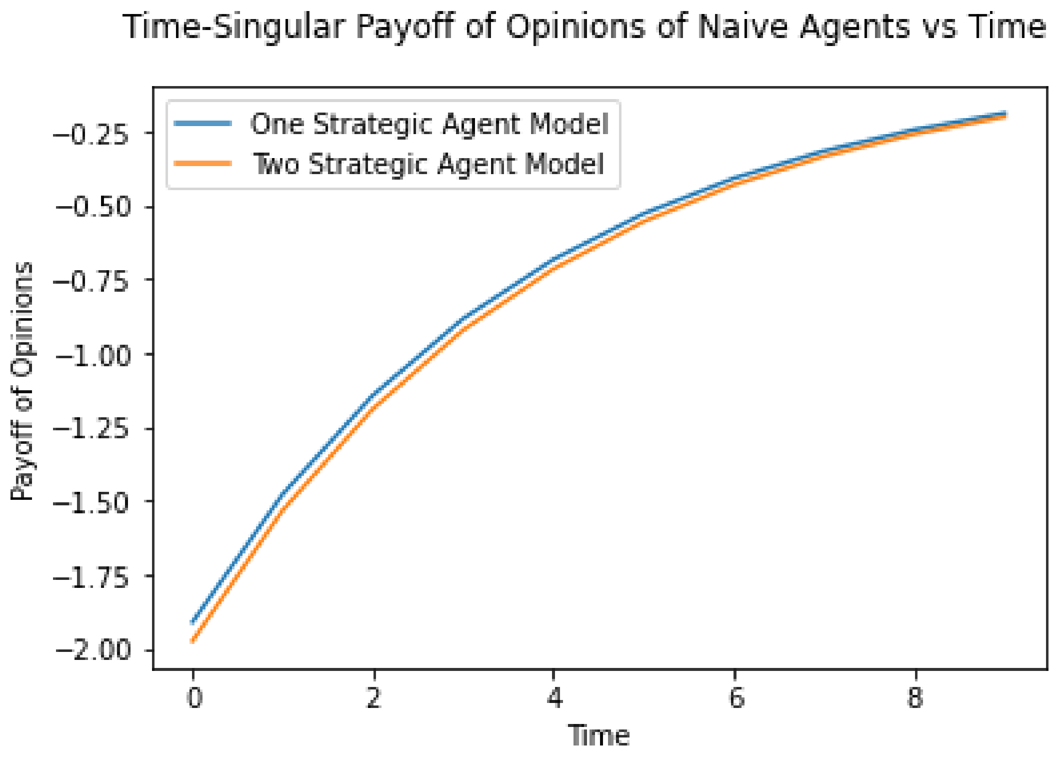
```
[31]: -20.69912623063974
```

In the two-agent model they are in fact lower. One would think they should be higher since the messages are less costly.

```
[32]: fig, ax = plt.subplots()
      fig.suptitle("Time-Singular Payoff of Opinions of Naive Agents vs Time")
      plt.plot(range(10), [-delta**i  * (x_i.T @ Q[0] @ x_i).item() for i, x_i in␣
       ↪enumerate(save_xs[0][:10])], label = "One Strategic Agent Model")
      plt.plot(range(10), [-delta**i  * (x_i.T @ Q[0] @ x_i).item() for i, x_i in␣
       ↪enumerate(xs[0][:10])], label = "Two Strategic Agent Model")
      plt.xlabel("Time")
      plt.ylabel("Payoff of Opinions")
      ax.legend()
      plt.show()
```

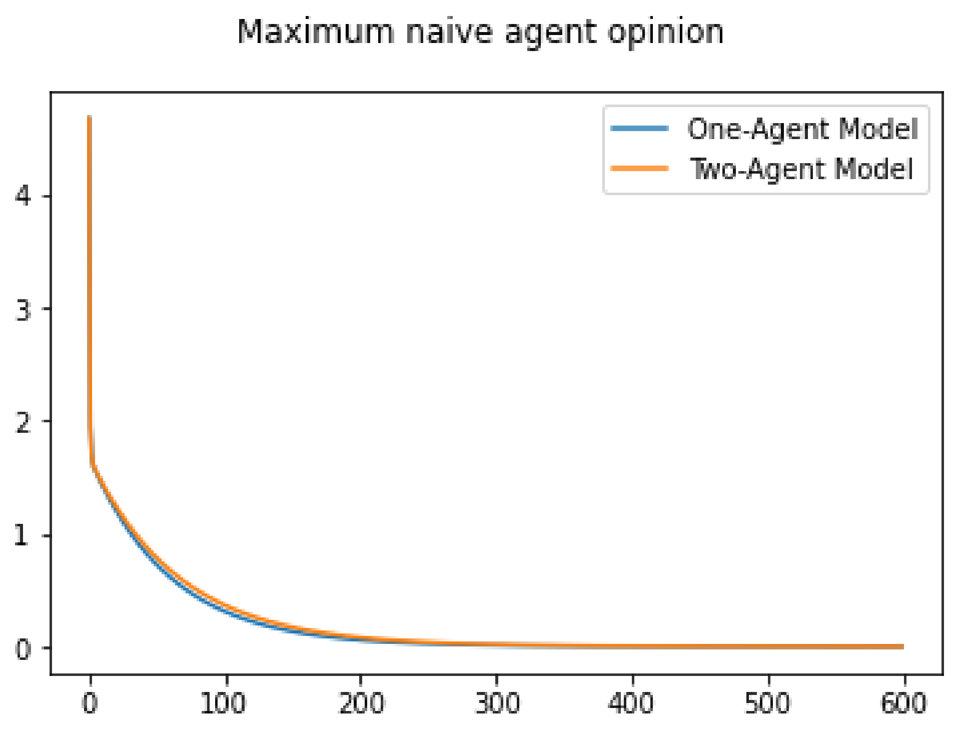


[33]:
```
fig, ax = plt.subplots()
fig.suptitle("Time-Singular Payoff of Opinions of Naive Agents vs Time")
plt.plot(range(10), [-delta**i  * (x_i.T @ Q[0] @ x_i).item() for i, x_i in␣
 ↪enumerate(save_xs[0][10:20])], label = "One Strategic Agent Model")
plt.plot(range(10), [-delta**i  * (x_i.T @ Q[0] @ x_i).item() for i, x_i in␣
 ↪enumerate(xs[0][10:20])], label = "Two Strategic Agent Model")
plt.xlabel("Time")
plt.ylabel("Payoff of Opinions")
ax.legend()
plt.show()
```



At a very small scale, the payoff is lower in the two-strategic model. This is what leads to the lower payoff in the long-run.

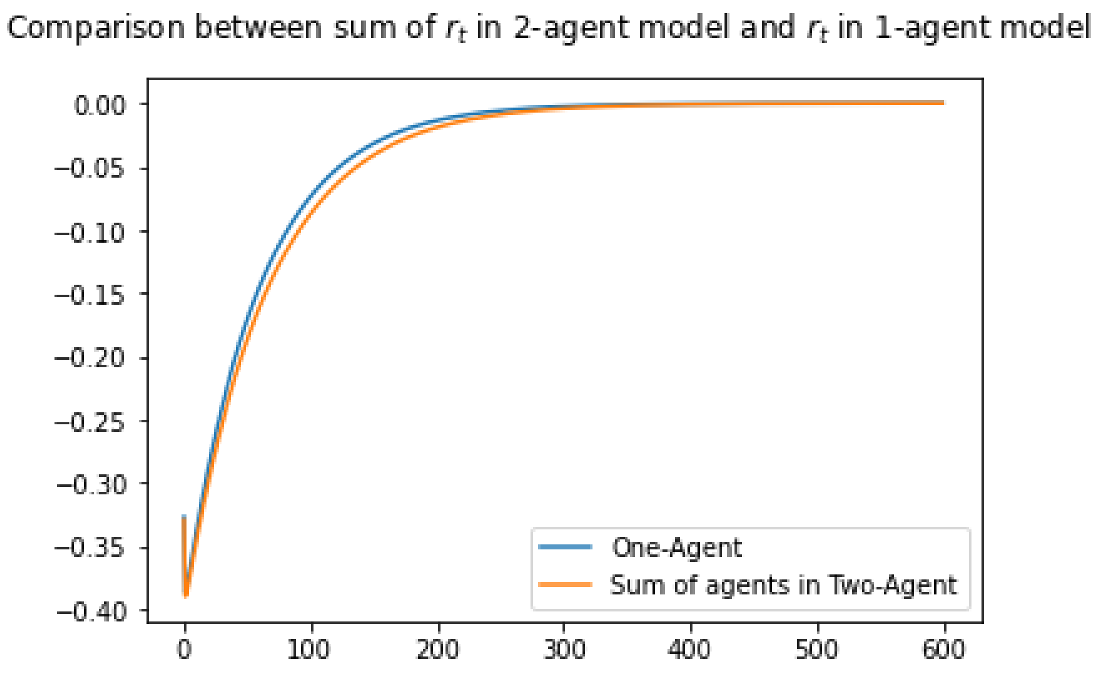
[34]:
```
fig, ax = plt.subplots()
fig.suptitle("Maximum naive agent opinion")
plt.plot(range(600), [np.max(i) for i in save_xs[0][:600]], label = "One-Agent␣
↪Model")
plt.plot(range(600), [np.max(i) for i in xs[0][:600]], label = "Two-Agent␣
↪Model")
ax.legend()
plt.show()
```



This plot shows that the naive agents have higher, greater-than-zero opinions in the two-agent model than in the one-agent model. This would impose a greater cost in the two-agent model long-term as everything else remains the same with respect to the opinions.

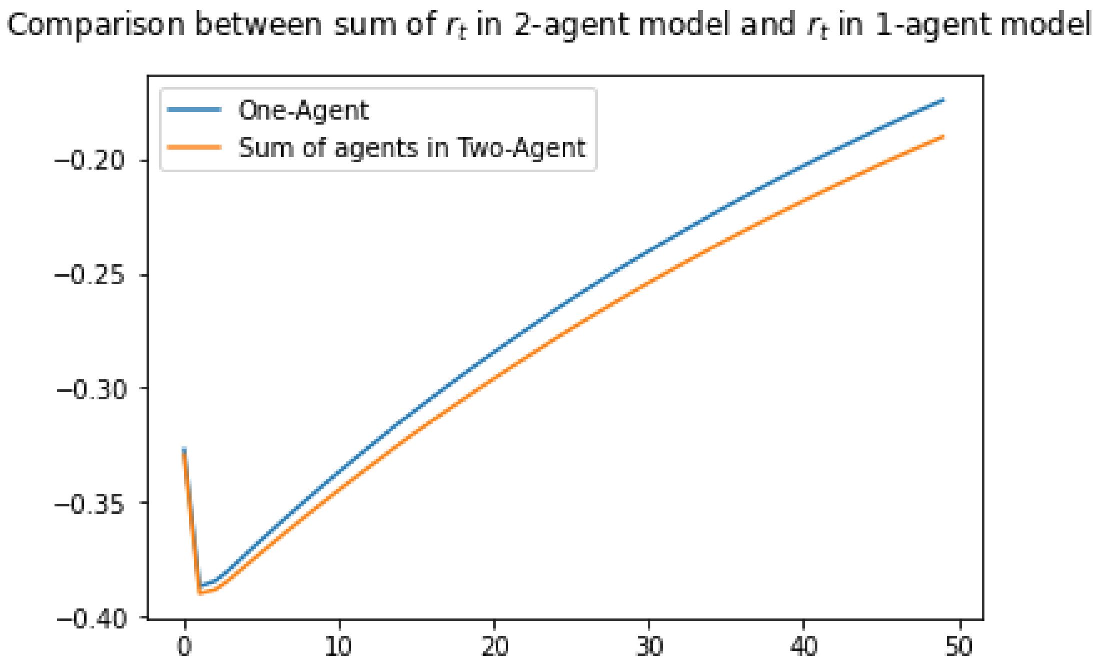
It makes sense that these opinions would be higher because the two strategic agents are sending less intense messages.

[35]:
```
fig, ax = plt.subplots()
fig.suptitle("Comparison between sum of $r_t$ in 2-agent model and $r_t$ in␣
 ↪1-agent model")
plt.plot(range(600), [rx.item() for rx in rs[2][:600]], label = "One-Agent")
plt.plot(range(600), [rx.item() + ry.item() for rx, ry in zip(rs[0][:600],␣
 ↪rs[1][:600])], label = "Sum of agents in Two-Agent")
ax.legend()
plt.show()
```

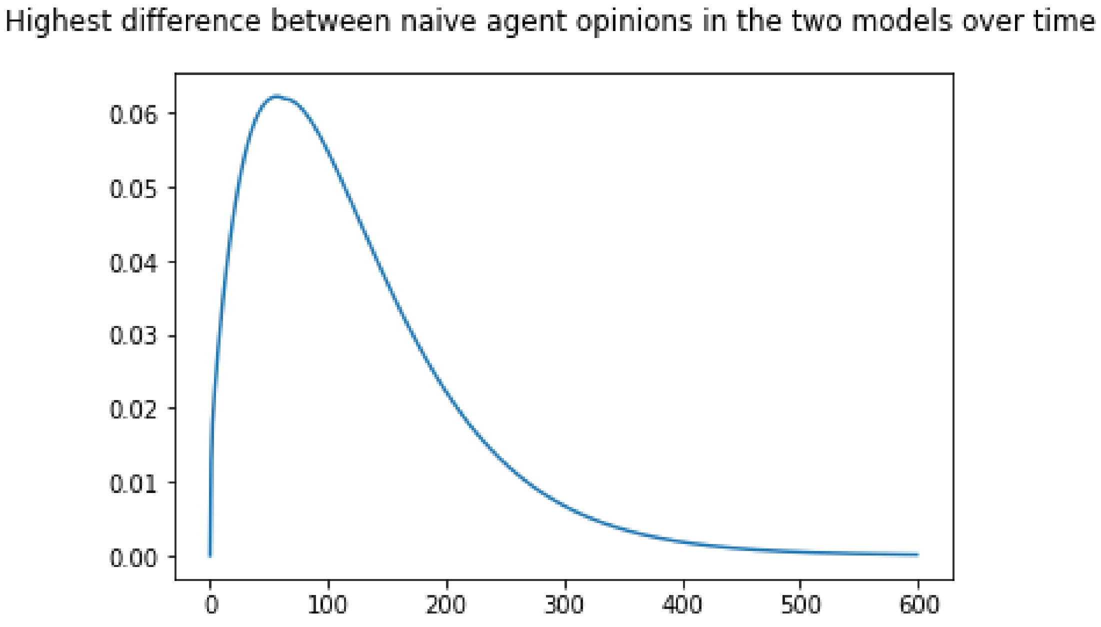


The sum of the messages is further away from zero in the two-agent model than in the one-agent model.

[36]:
```
fig, ax = plt.subplots()
fig.suptitle("Comparison between sum of $r_t$ in 2-agent model and $r_t$ in␣
 ↪1-agent model")
plt.plot(range(50), [rx.item() for rx in rs[2][:50]], label = "One-Agent")
plt.plot(range(50), [rx.item() + ry.item() for rx, ry in zip(rs[0][:50], rs[1][:
 ↪50])], label = "Sum of agents in Two-Agent")
ax.legend()
plt.show()
```

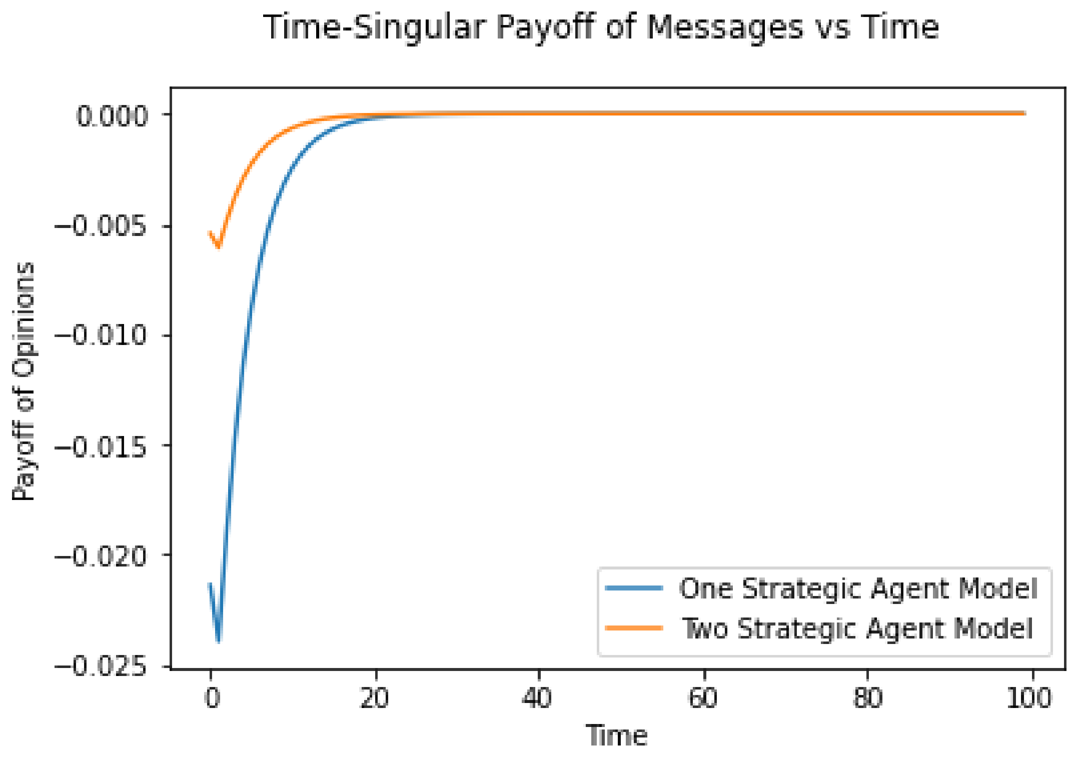


[37]:
```
fig, ax = plt.subplots()
fig.suptitle("Highest difference between naive agent opinions in the two models␣
 ↪over time")
plt.plot(range(600), [np.max(i - j) for i, j in zip(xs[0][:600], save_xs[0][:
 ↪600])])
plt.show()
```



This plot shows that the naive agents in the two-agent model have overall higher opinions in the short-run than those in the one-agent model. This may contribute to a worse long-run payoff.

[38]:
```
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()
```



This is as expected, the messages for the agents in the two-agent model are less costly than that of the one-agent model. But notice how small the payoff factor actually is; it is negligible compared to the payoff factor of the opinions.

### 1.4.3 This was a realization I had later:

- The network was split by 0.03955 fraction of influence to the two strategic agents

- Suppose we had $r_k$ in the one-strategic-agent case. Then the movement on the network would be 0.0791 * $r_k$.
- Now, suppose we had the two-strategic-agent network. 0.03955 * $r_k$ + 0.03955 * $r_k$ = 0.0791 * $r_k$.
- Alternatively, 0.0791 * $r_k/2$ + 0.0791 * $r_k/2$ = 0.0791 * $r_k$.
- However, in our model, we have something along the lines of 0.03955 * $r_l$ + 0.03955 * $r_l$ where $r_l > r_k$. This produces a number greater than 0.0791 * $r_k$, which is bad because the intention is for the number to be more negative.
- While $r_l > r_k$, it is also the case that $r_l + r_l < r_k$.

The end result is that $r_l$ of the strategic agent in the 2-strategic-agent model needs to be equal to $r_k$ to get the same situation as the one-agent model. This does not happen. Instead, a higher $r_l$ is used (which still when summed up is more powerful than the original $r_k$, which causes a less powerful influence, which leads to the naive agents having opinions further away than before.

In summary, the way the network was split still requires $r_k$ to be the same to get the same opinion trajectory. This of course does not happen, and the two strategic agents use less intensive, cheaper messages which still converge to zero in the long run.

This is not socially efficient, because if the original $r_k$ was used, they would reflect the same system and payoff would be higher. Observe we currently have:

[39]:
```
payoff = 0
current_opinions = X_0_1
for i in range(len(rs[0])):
    payoff += -delta**i  * (current_opinions.T @ Q[0] @ current_opinions).
 ↪item() - delta**i  * (rs[0][i].T @ R[0] @ rs[0][i]).item()
    current_opinions = A @ current_opinions + 2 * B_1 @ rs[0][i]
payoff
```

[39]:
```
-20.699126230639745
```

But if we used the same $r_t$ sequence as the one-strategic-agent model for both strategic agents:

[40]:
```
payoff = 0
current_opinions = X_0_1
for i in range(len(save_rs[0])):
    payoff += -delta**i  * (current_opinions.T @ Q[0] @ current_opinions).
 ↪item() - delta**i  * (save_rs[0][i].T @ R[0] @ save_rs[0][i]).item()
    current_opinions = A @ current_opinions + 2 * B_1 @ save_rs[0][i]
payoff
```

[40]:
```
-20.666612675948908
```

The payoff becomes exactly the same as the one-strategic-agent model because the `2 * B_1 @ save_rs[0][i]` term is equivalent to `B @ r_t` from the one-agent model now.

In the Nash equilibrium, the strategy is not the same as the socially optimal strategy likely because were one of the agents to perform this socially optimal strategy, the other would be able to achieve a higher payoff by doing less work (because the network still converges even with less intensive messages as demonstrated in the Nash equilibrium)

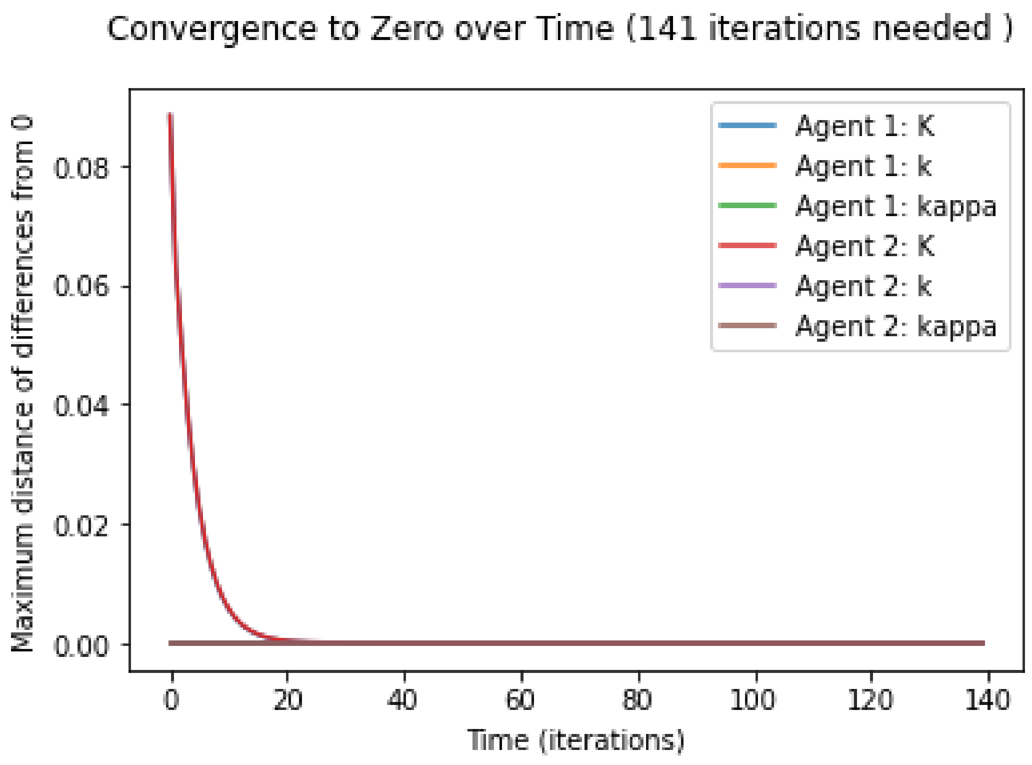
## 1.5 Clean Summary of Findings for Test 4 and Test 5:

- When two strategic agents are identical, given they split influence to one of the agents, there are two impacts on cost:
  - The payoff from the opinions of the naive agents are lower (more costly) in the short run, and the same in the long run. This is a result of the opinions being further away from zero in the two-agent model than in the one-agent model, which induces extra cost.
    - The opinions are further away because of the network structure as I outlined above, where the Nash equilibrium is worse than the socially optimal strategy - both agents want to minimize costs, but in factoring the opposing agent do worse off than the one-agent model (in preventing cost from getting even worse)
  - The payoff from the messages is higher (less costly) in the short run because of the less intensive (closer to zero) messages.
  - Because the numbers here are far smaller for the message payoff weights relative to the opinion weights, this higher message payoff only has an effect in the short-run. In the long-run, because of the higher weight on the opinions, and because the opinions are the same or higher than before, the cumulative payoff in the two-agent case is lower (more costly) than that of the one-agent case in the long run. This would not be the situation if the weight of the messages were more than the weight of the opinions, however.

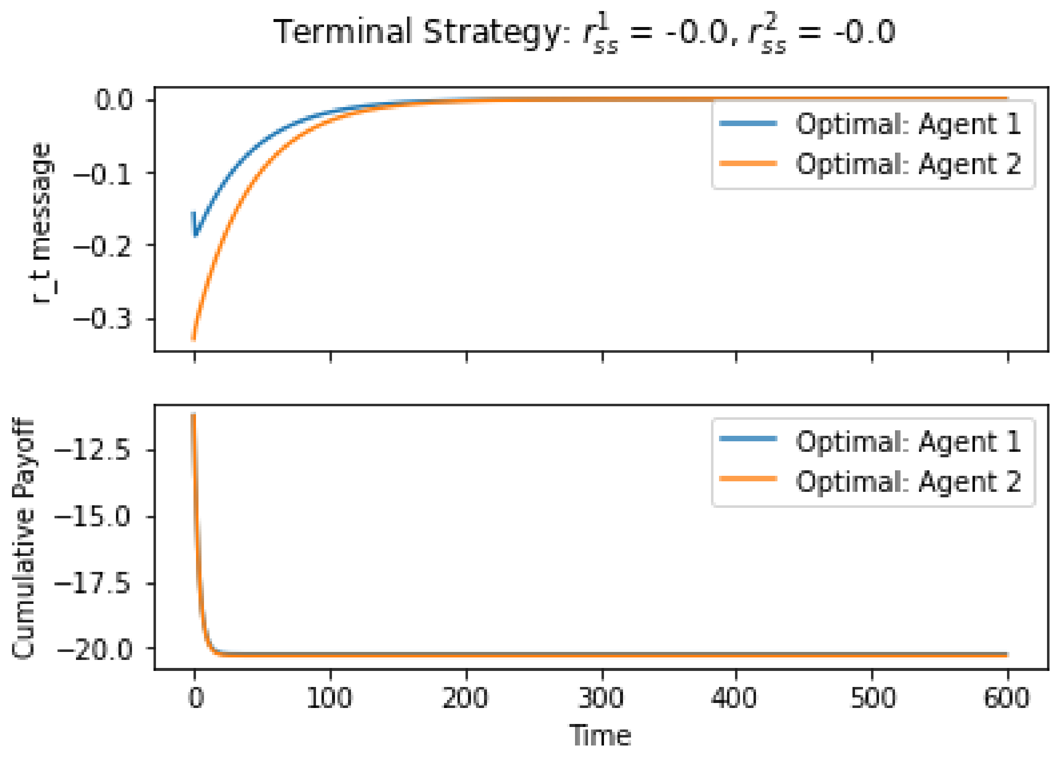
## 1.6 Alternate Setup: Asymmetric Influence

[41]:
```
A = np.array([
  [0.217 + 0.03955/5,    0.2022 + 0.03955/5,    0.2358 + 0.03955/5,    0.1256 + 0.
↪03955/5,    0.1403 + 0.03955/5], # uniformly increase here by 0.03955 / 5
  [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 - 0.03955/5,    0.0711 - 0.03955/5,    0.0253 - 0.03955/5,    0.2244 - 0.
↪03955/5,    0.5536 - 0.03955/5], # uniformly decreaase here by 0.03955 / 5
], ndmin = 2)

B_1 = np.array([
  0.03955,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B_2 = np.array([
  0,
  0,
  0,
  0,
  0.03955,
], ndmin = 2).T

B = [B_1, B_2]

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)]
```

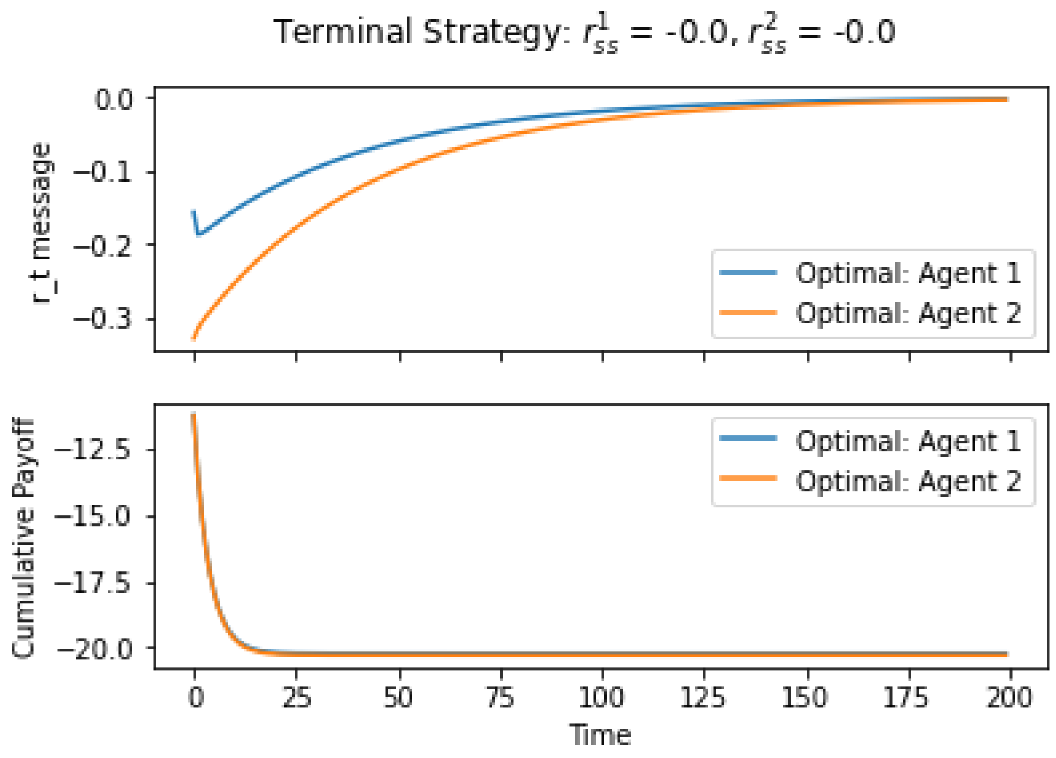
[42]:
```
max_distances, historical_K, historical_k, historical_kappa = solve(Q, [np.
 ↪zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x, c,␣
 ↪tol = 1000)
converge_plot(max_distances, tol = 1000)
```



[43]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, set_cap = 600)
```



[44]:
```
do_plot(rs, r, payoffs, set_cap = 200)
```



The payoffs are similar because again, the payoff from the messages is negligible compared to that of the opinions, which are the same between agents. But here, Strategic Agent 1 which influences naive agent 1, has less intensive messages (and would therefore see better payoff than Strategic Agent 2).

[45]:
```
(- X_0_1.T @ historical_K[0][0] @ X_0_1 - historical_k[0][0] @ X_0_1 +␣
↪historical_kappa[0][0]).item() # STRATEGIC AGENT 1
```

[45]:
```
-20.23209924567731
```

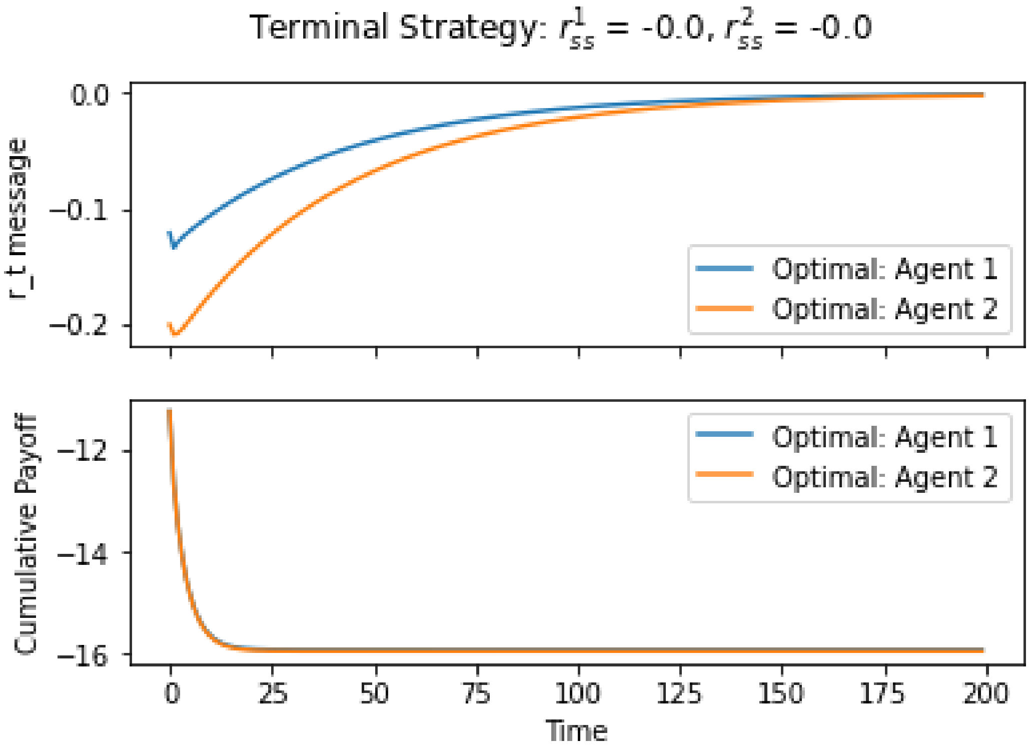
[46]:
```
(- X_0_1.T @ historical_K[0][1] @ X_0_1 - historical_k[0][1] @ X_0_1 +␣
↪historical_kappa[0][1]).item() # STRATEGIC AGENT 2
```

[46]:
```
-20.291191278569375
```

## 1.7 Alternate Setup 2: switch initial opinions of naive agent 1 and 5

[47]:
```
A = np.array([
  [0.217 + 0.03955/5,    0.2022 + 0.03955/5,   0.2358 + 0.03955/5,   0.1256 + 0.
↪03955/5,   0.1403 + 0.03955/5], # uniformly increase here by 0.03955 / 5
  [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 - 0.03955/5,   0.0711 - 0.03955/5,   0.0253 - 0.03955/5,   0.2244 - 0.
↪03955/5,   0.5536 - 0.03955/5], # uniformly decreaase here by 0.03955 / 5
], ndmin = 2)

B_1 = np.array([
  0.03955,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B_2 = np.array([
  0,
  0,
  0,
  0,
  0.03955,
], ndmin = 2).T

B = [B_1, B_2]

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

[48]:
```
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)
do_plot(rs, r, payoffs, set_cap = 200)
```



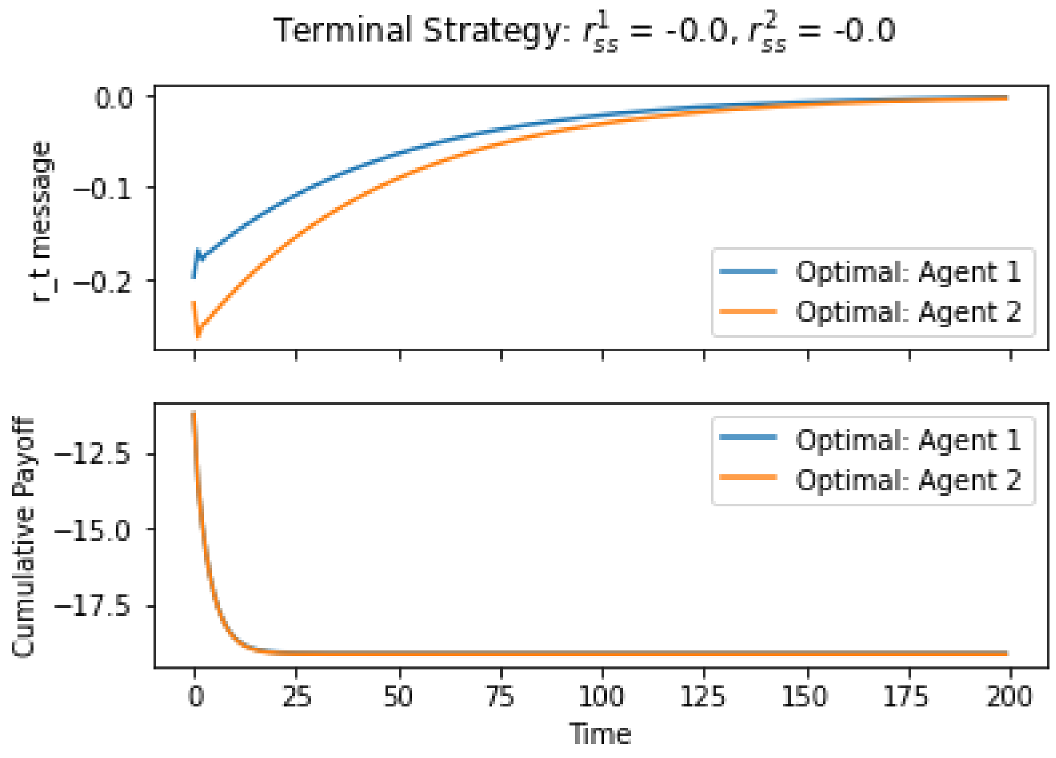
This did not lead to a swap in strategy levels.

## 1.8 Alternate Setup 3: swap the network weights instead

[49]:
```
A = np.array([
  [0.1256 - 0.03955/5,   0.0711 - 0.03955/5,   0.0253 - 0.03955/5,   0.2244 - 0.
↪03955/5,   0.5536 - 0.03955/5], # SWAPPED
  [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],
  [0.217 + 0.03955/5,    0.2022 + 0.03955/5,   0.2358 + 0.03955/5,   0.1256 + 0.
↪03955/5,   0.1403 + 0.03955/5], # uniformly increase here by 0.03955 / 5
], ndmin = 2)

B_1 = np.array([
  0.03955,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B_2 = np.array([
  0,
  0,
  0,
  0,
  0.03955,
], ndmin = 2).T

B = [B_1, B_2]

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)]
```

[50]:
```
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)
do_plot(rs, r, payoffs, set_cap = 200)
```



This did not swap them either.