MultipleInfluence

July 19, 2021

1 Programmatic Engine for Calculation of Optimal Strategies in Competitive Influence Model

James Yu, 18 July 2021

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

In this notebook I construct a set of parameter-based functions capable of solving the competing influencer model for any choice of initial states, conditional on R being a 1 by 1 matrix. Modifications are required to support additional channels.

The key equations needed to solve the model are the three recursive equations for K^l_{t-1}, k^l_{t-1} and κ^l_{t-1}. To obtain those, functions are needed for supporting matrices used in those equations:

[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_

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]
```

This allows the recursive equations to be represented as follows:

[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)]
```

1.1 Test #1: The Original 5-Agent Model (no bot)

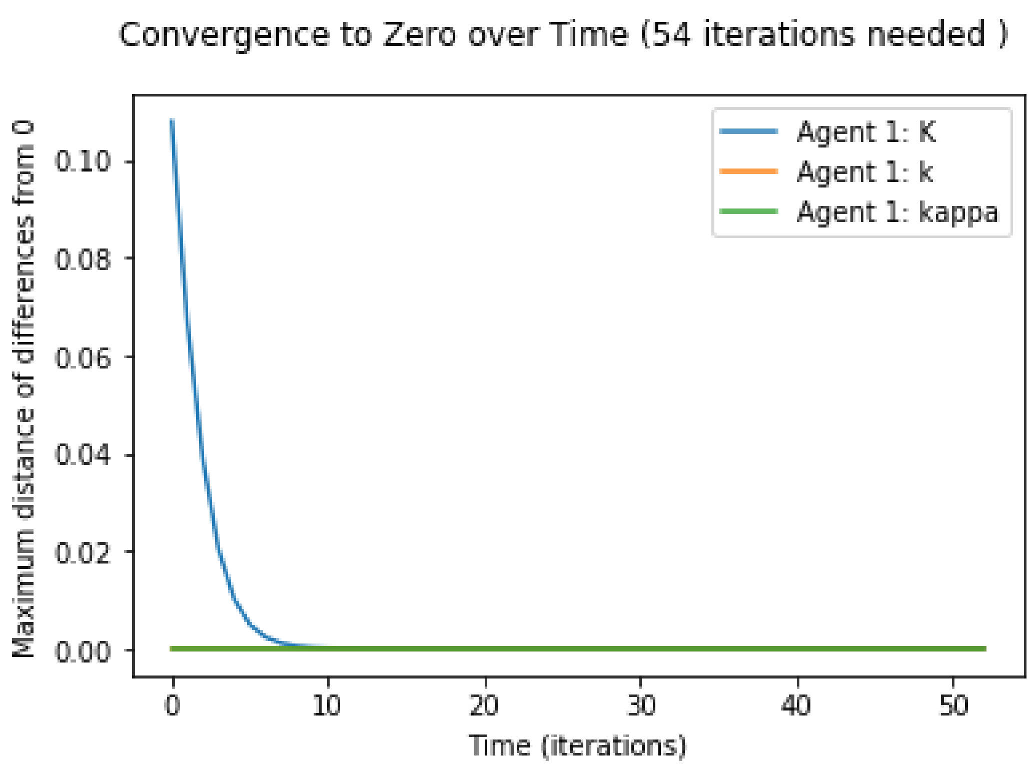
[4]:
```
A = np.array([
   [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
   [0.2497,   0.0107,   0.2334,   0.1282,   0.378],
   [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
   [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
   [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
], ndmin = 2)
```

```
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)]
```

[5]:
```
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

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)
```

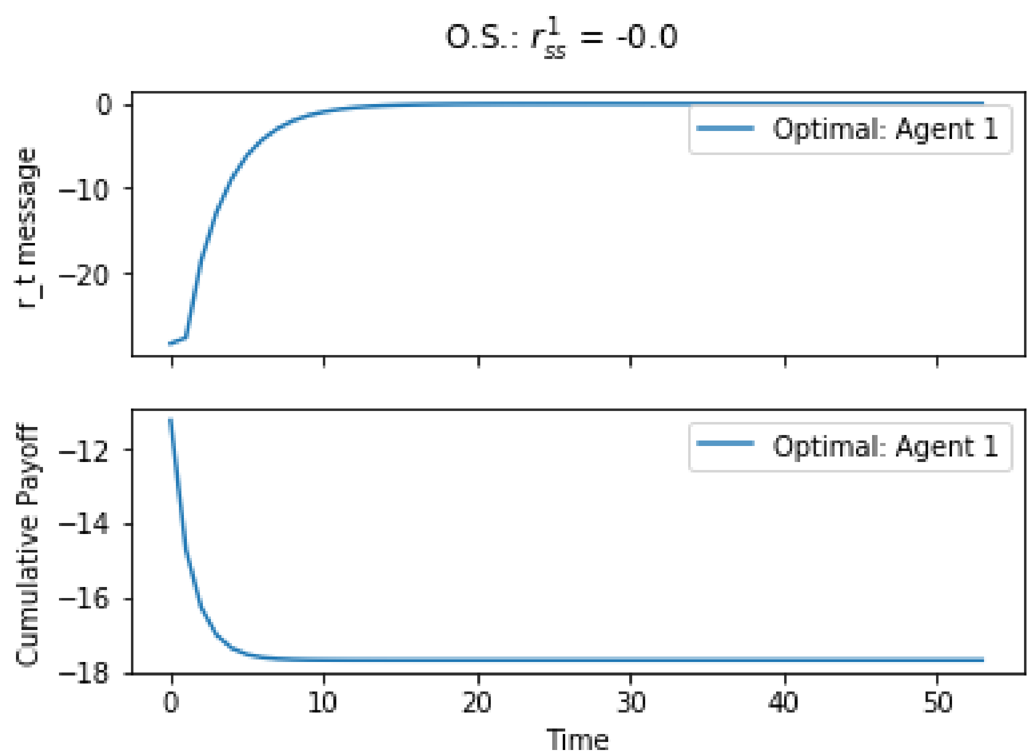
[6]:
```
def converge_plot(max_distances, tol = 300):
    fig, ax = plt.subplots()
    fig.suptitle(f"Convergence to Zero over Time ({len(max_distances[(1, 'K')])␣
 ↪+ 1} iterations needed {['', '- rounding error␣
 ↪observed'][len(max_distances[(1, 'K')]) + 1 == tol + 2]})")
    for l in max_distances:
        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()

converge_plot(max_distances)
```



Note that it gets quite close to zero at about 10 iterations.
To obtain the optimal strategy, the following is used:

[7]:
```
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]

def optimal(X_init, historical_K, historical_k, historical_kappa):
    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)
    for i in range(len(historical_K)):
        K_t = historical_K[i]
        k_t = historical_k[i]
        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)
        for l in range(L):
            Y_new = rs[l][-1]
            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)
            X_t[l] = X_new

    return xs, rs, payoffs

xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
```

[8]:
```python
def do_plot(rs, r, payoffs):
    fig, sub = plt.subplots(2, sharex=True)
    fig.suptitle(f"O.S.: {', '.join(['$r_{ss}^' + str(l+1) + '$ = ' +␣
 ↪str(round(rs[l][-1].item() + r[l], 2)) for l in range(L)])}")

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

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

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

do_plot(rs, r, payoffs)
```

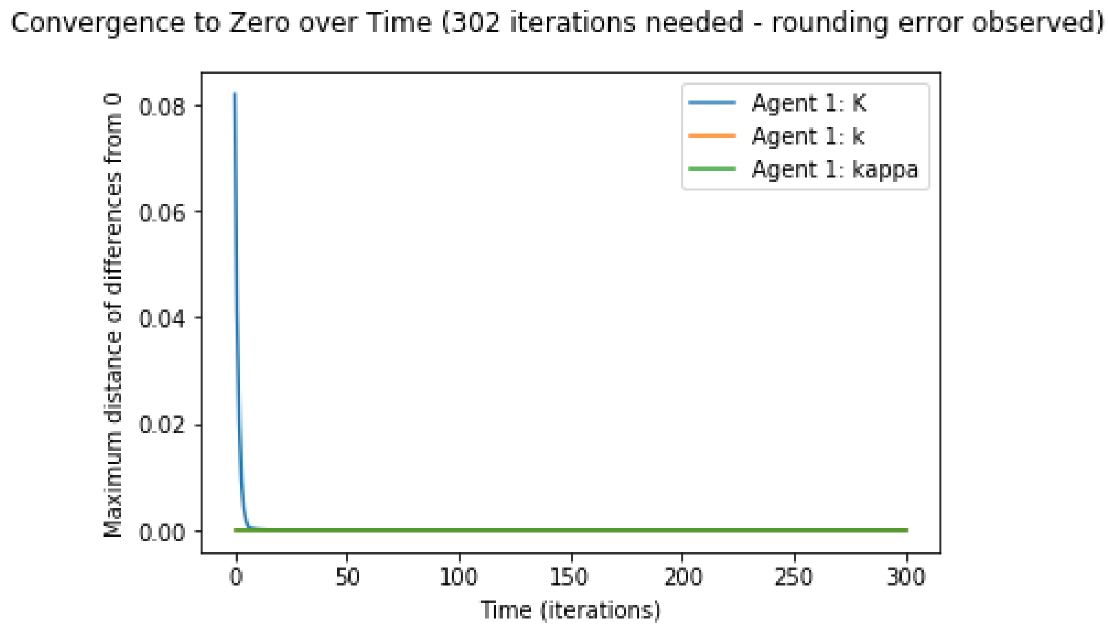


This graph is identical to that of the original setup, proving that at least this component of the engine functions properly.

1.2 Test #2: The Original 5-Agent Model (with bot)

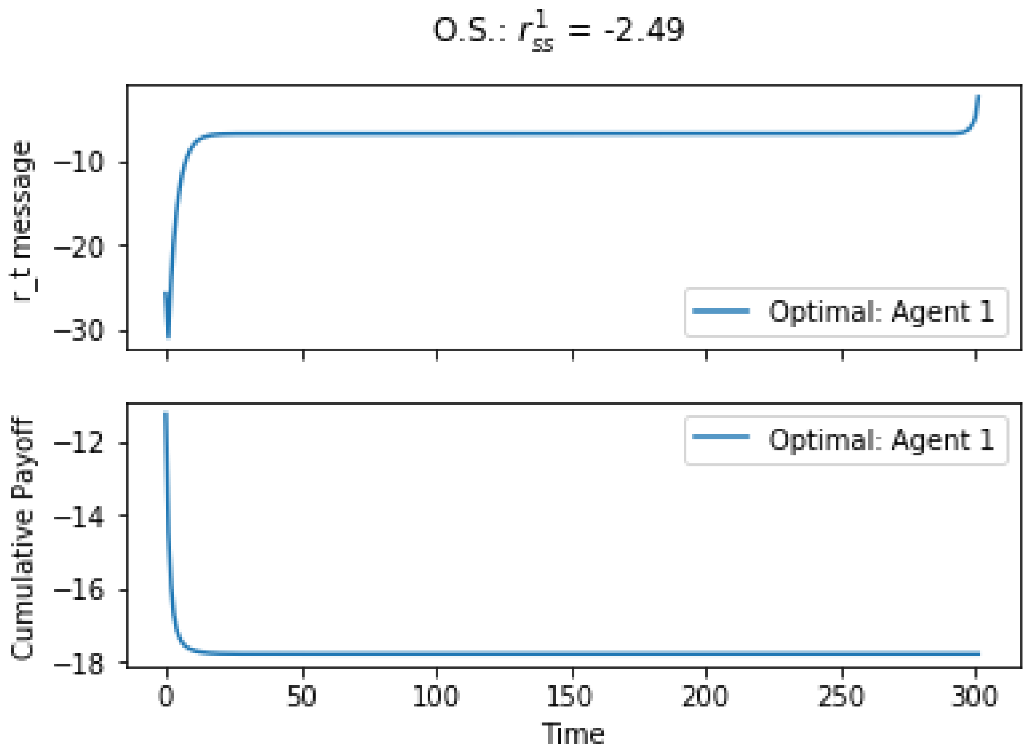
[9]:
```
A = np.array([
  [1,        0,        0,        0,        0,        0     ], # the bot
  [0,        0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.1012,   0.8988*0.2497,   0.8988*0.0107,   0.8988*0.2334,   0.8988*0.1282,
 ↪ 0.8988*0.378 ],
  [0,        0.1285,   0.0907,   0.3185,   0.2507,   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]
```

[10]:
```
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)
```



(note that this does not actually converge perfectly to zero due to rounding error (and the 300 is a forced cutoff, NOT THE ACTUAL CUTOFF); from the convergence plot however, one can see it does get close)

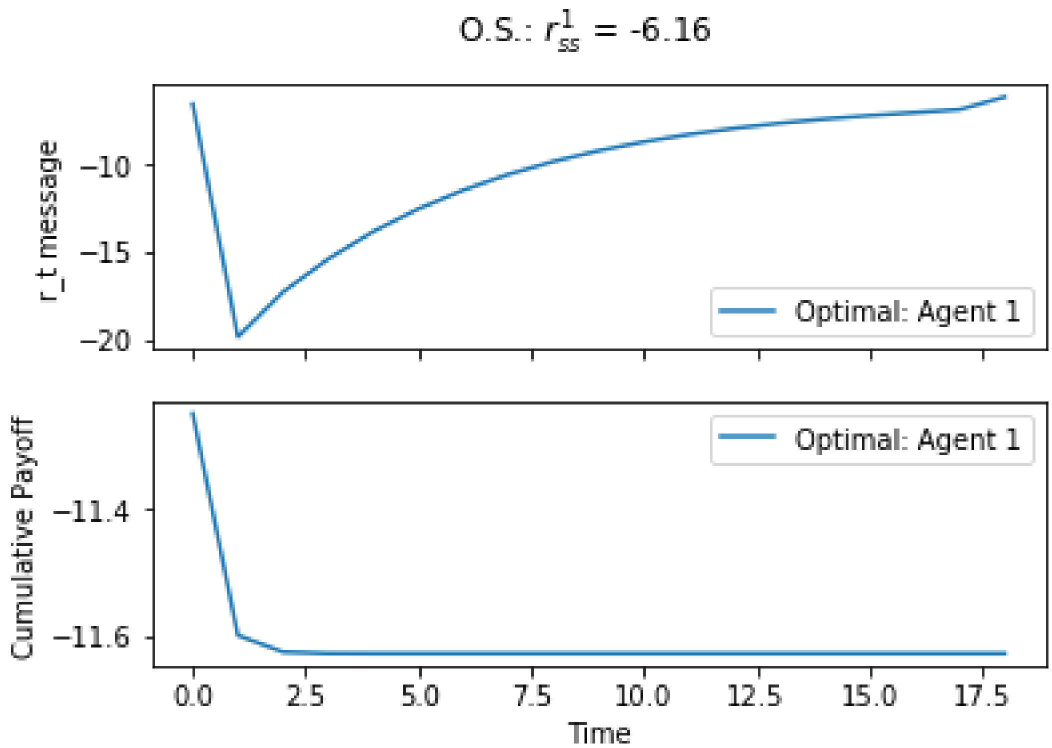
[11]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs)
```



Again, THIS IS NOT THE ACTUAL r_{ss}^1 because of the tail problem that comes from a prior experiment. But this proves that the engine is working as intended in this regard.

1.3 Test #3: Ensure delta Parameter Variation Works as Intended

[12]:
```
delta = 0.1
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)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs)
```



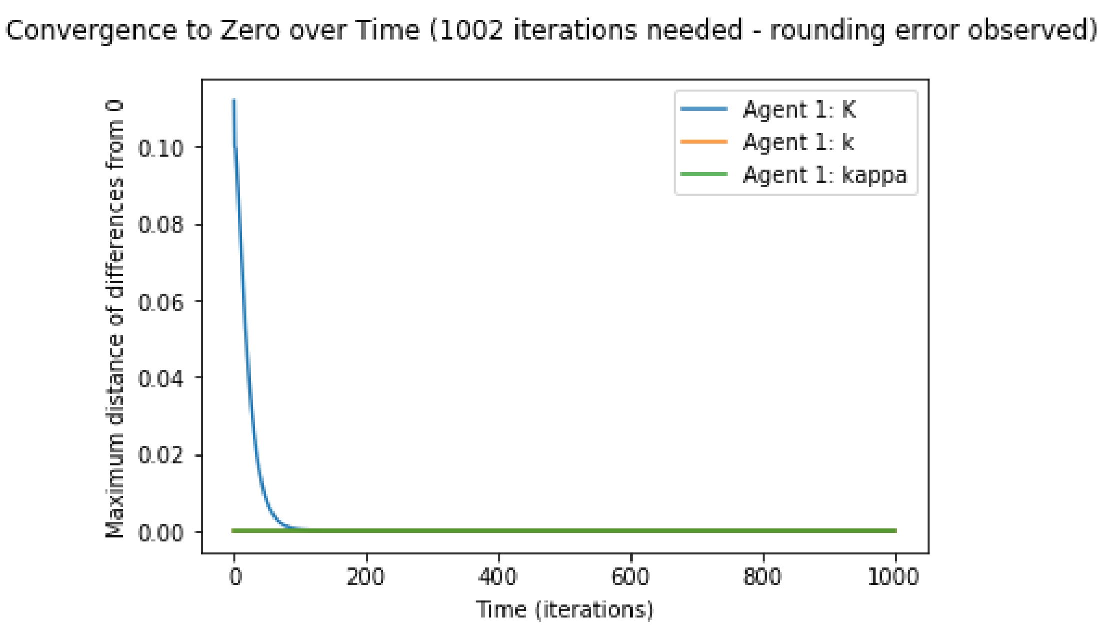
Behaviour is as expected.

1.4 Test #4: Single Strategic Agent, 5-agent Setup, Induce Cost (benchmark for a later test)

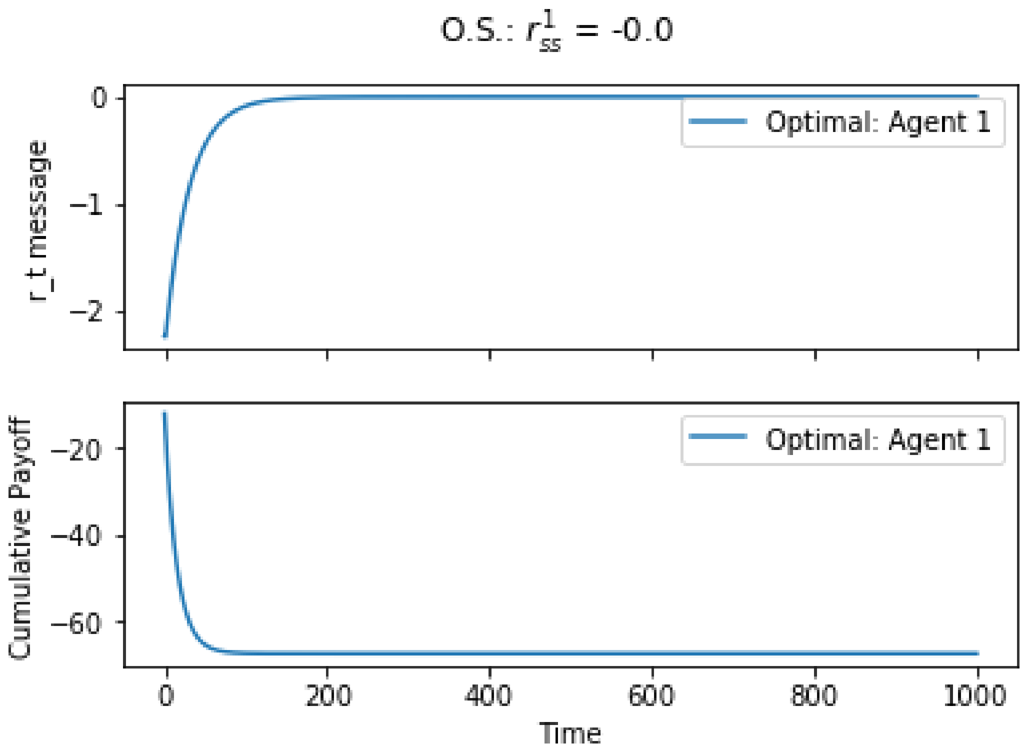
[13]:
```
A = np.array([
  [0.217,    0.2022,   0.2358,   0.1256,   0.1403],
  [0.2497,   0.0107,   0.2334,   0.1282,   0.378],
  [0.1285,   0.0907,   0.3185,   0.2507,   0.2116],
  [0.1975,   0.0629,   0.2863,   0.2396,   0.2137],
  [0.1256,   0.0711,   0.0253,   0.2244,   0.5536],
], ndmin = 2)

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 = 1
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)]
```

[14]:
```
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)
```



[15]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs)
```



[16]:
```
save_variable = rs
save_payoff = payoffs[0][-1]
```

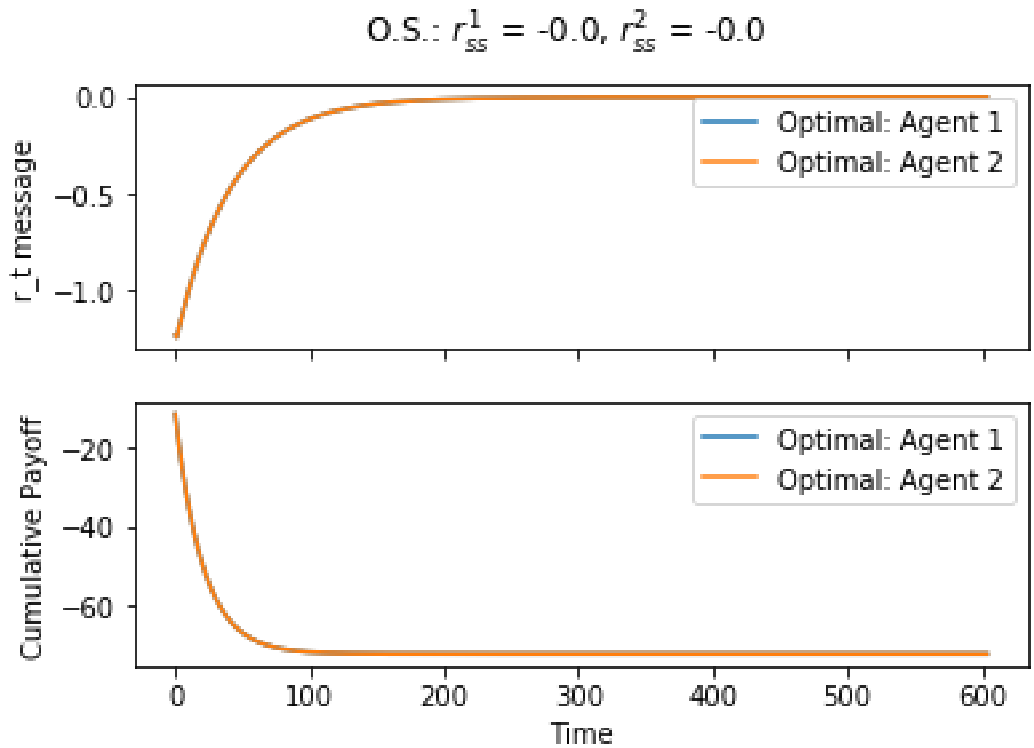
1.5 Test #5: Dual Strategic Agent, Equal Agendas, Split Influence to Agent 1

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

B = [B_1, B_1]
```

```python
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 = 1
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)]
```

[18]:
```python
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)
```

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



Note that the payoffs are lower and the messages are higher. As the agents are identical, the strategies are identical. Also note that because the agents are identical, the strategy is close to (but not exactly) half of what it was before.

[20]:
```
print(save_payoff, " is the original payoff")
print(payoffs[0][-1], " is the new payoff")
```

```
-67.32909534366108  is the original payoff
-72.13273885939068  is the new payoff
```

[21]:
```
max(abs(sum([np.array([a.item() for a in rs[l]]) for l in range(L)]) - np.
↪array([a.item() for a in save_variable[0]])[0:605]))
```

[21]:
```
0.3736385551591217
```

This is the maximum difference between the sum of the r_t strategies in this case and those of the prior case. It is relatively small.

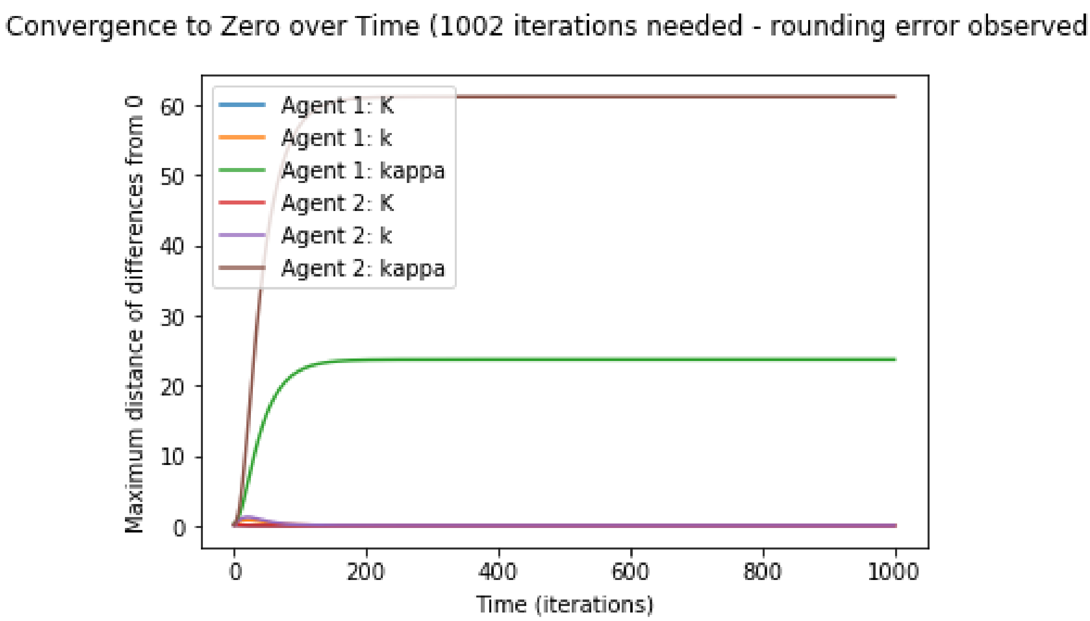
1.6 Experiment 1: Have one of the Strategic Agents Take Over the role of the Bot

[22]:
```
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)

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

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], 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_2 = np.array([
  -0.98 - 10,
  -4.62 - 10,
  2.74  - 10,
  4.67  - 10,
  2.15  - 10,
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 1
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 10]
r = [0, 10]
```

```
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, [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)
```



[24]:
```
for key in max_distances:
    print(max_distances[key][-1])
```

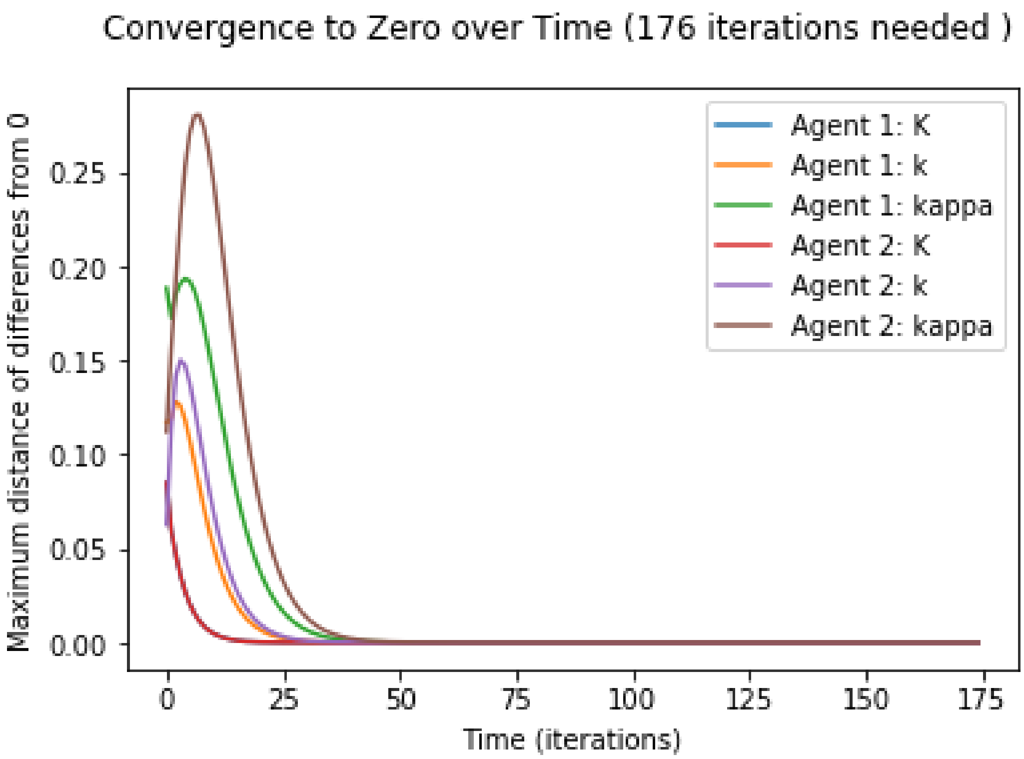
```
0.0
2.1316282072803006e-14
23.689564979493298
0.0
2.1316282072803006e-14
61.19221008454042
```

This does not converge. If instead we set $\delta = 0.8$, the following happens:

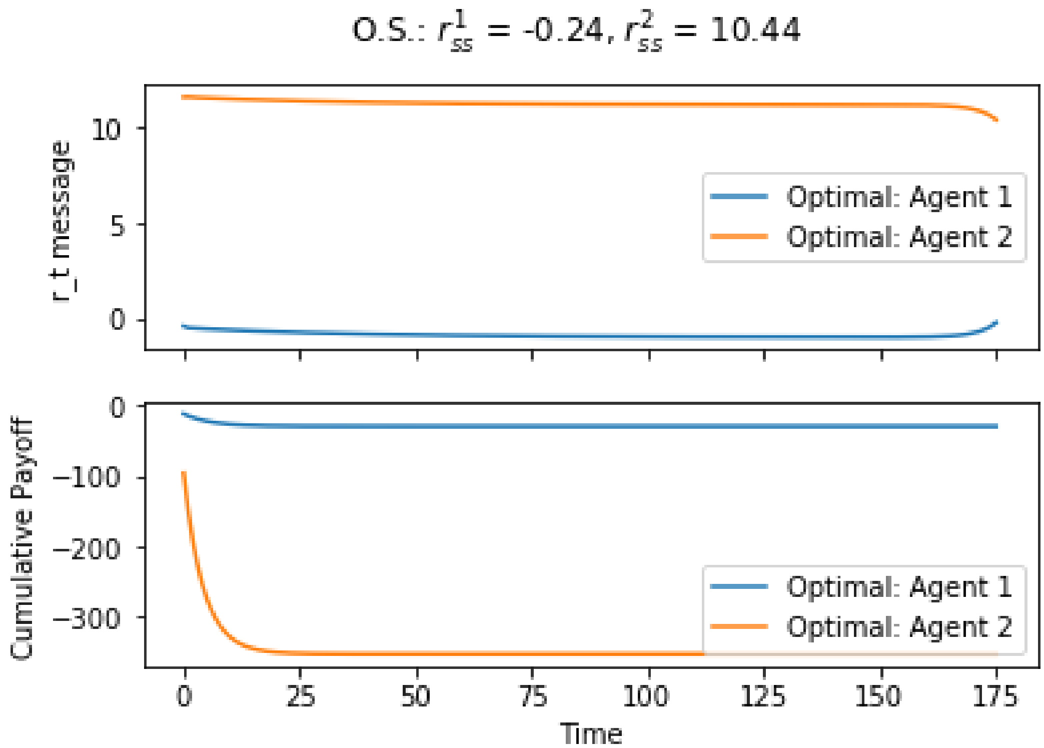
[25]:
```
A = np.array([
  [0.217,    0.2022,    0.2358,    0.1256,    0.1403],
  [0.8988*0.2497,    0.8988*0.0107,    0.8988*0.2334,    0.8988*0.1282,    0.8988*0.
 ↪378],
  [0.1285,    0.0907,    0.3185,    0.2507,    0.2116],
  [0.1975,    0.0629,    0.2863,    0.2396,    0.2137],
  [0.1256,    0.0711,    0.0253,    0.2244,    0.5536],
], ndmin = 2)
```

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

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], 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_2 = np.array([
  -0.98 - 10,
  -4.62 - 10,
  2.74  - 10,
  4.67  - 10,
  2.15  - 10,
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 10]
r = [0, 10]
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)]
```

```
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)
```



[26]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs)
```



[27]:
```
payoffs[0][-1]
```

[27]:
```
-28.983080789340928
```

There is a lot to unpack here. First of all, r_{ss} of both agents converges to close to their respective agendas. Second, the payoff of agent 2 is far lower than that of agent 1.

Most curious is the behaviour of agent 2. First of all, the opinions of the naive agents at convergence are:

[28]:
```
xs[0][-1]
```

[28]:
```
array([[3.70265674],
       [4.57049613],
       [3.99167739],
       [3.95464891],
       [3.97667971]])
```

[29]:
```
xs[1][-1] + (10 * np.ones((5, 1)))
```

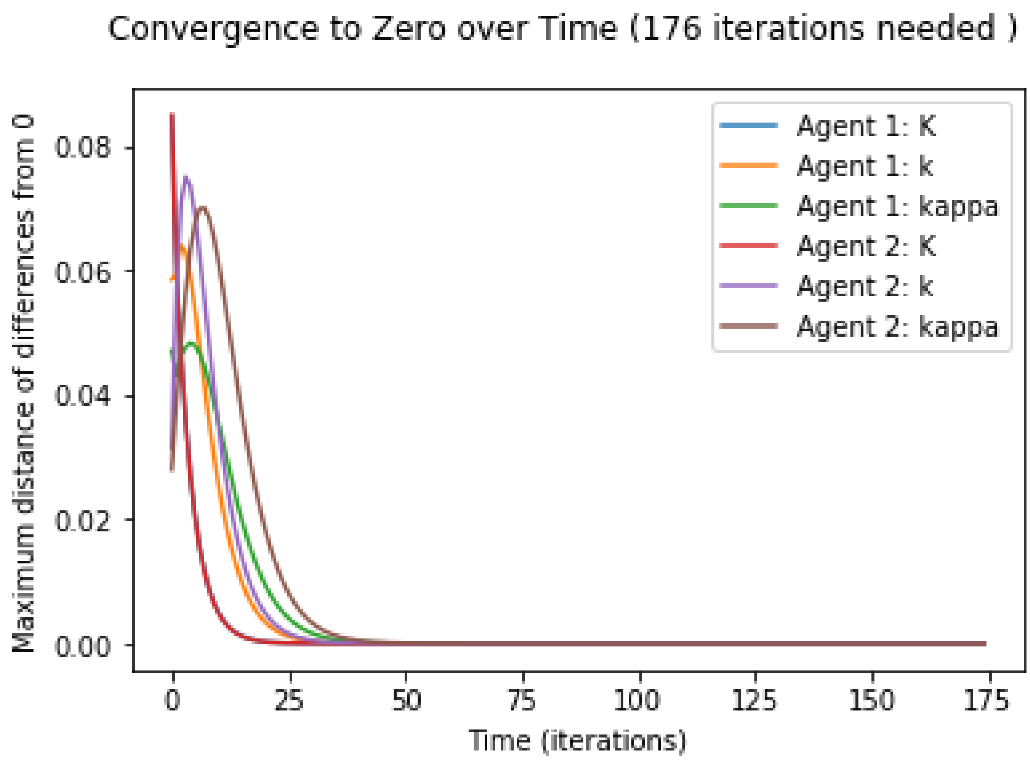
[29]:
```
array([[3.70265674],
       [4.57049613],
       [3.99167739],
       [3.95464891],
       [3.97667971]])
```

This verifies that they are the same in the end for each strategic agent's view. Also note they are biased more toward agent 2 (bot replacement) than agent 1 compared to prior examples.

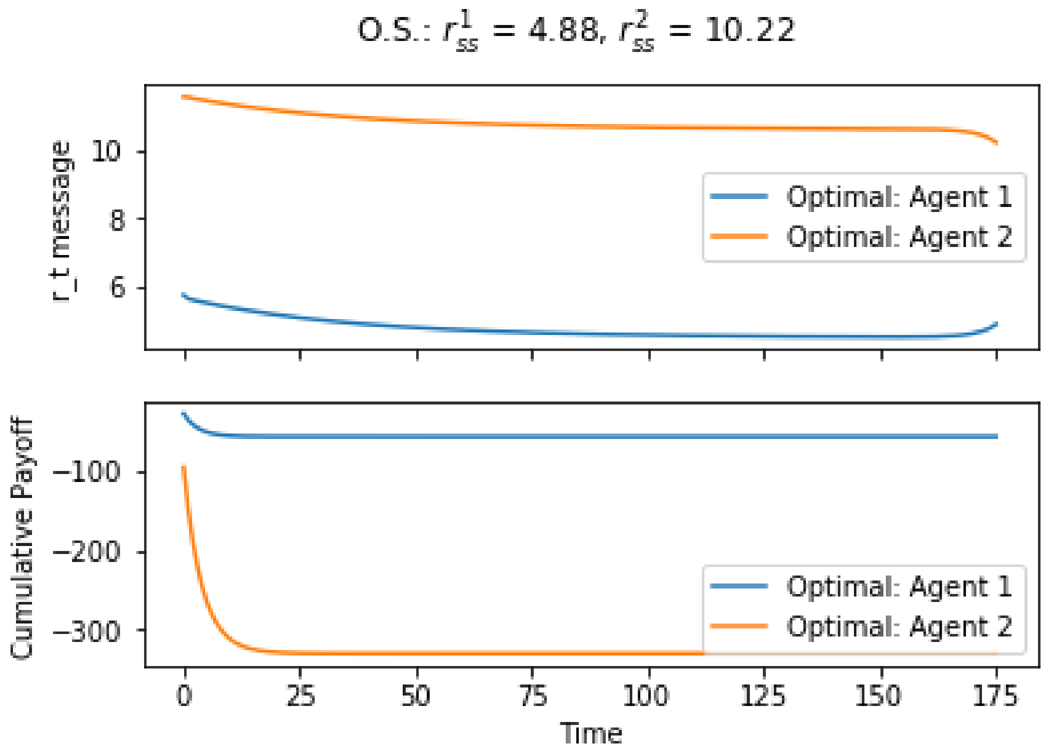
2 Experiment 2: Vary Agent 1's Agenda (to check impact on Agent 2)

[30]:
```
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)

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

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_2]

X_0_b = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T

X_0 = [X_0_b - (5 * np.ones((5, 1))), X_0_b - (10 * np.ones((5, 1)))]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [5, 10]
```

```
r = [5, 10]
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)]

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)
```



[31]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs)
```



[32]:
```
xs[0][-1] + (5 * np.ones((5, 1)))
```

[32]:
```
array([[6.82163157],
       [7.2556472 ],
       [6.96252772],
       [6.94416773],
       [6.95488608]])
```

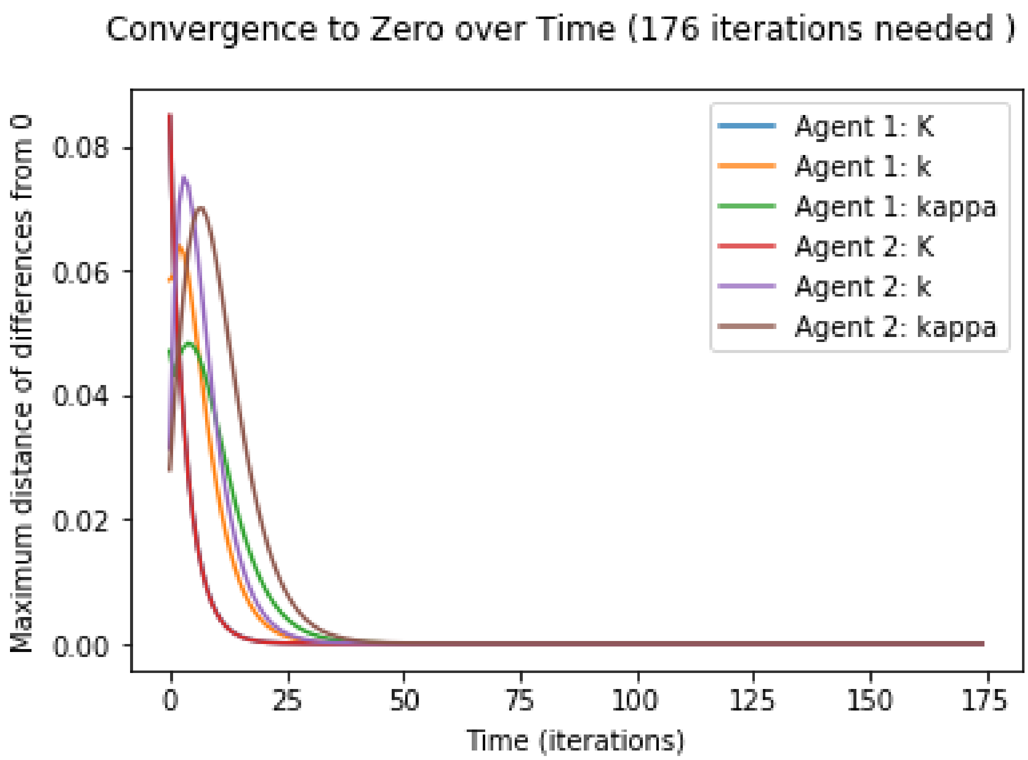
[33]:
```
xs[1][-1] + (10 * np.ones((5, 1)))
```

[33]:
```
array([[6.82163157],
       [7.2556472 ],
       [6.96252772],
       [6.94416773],
       [6.95488608]])
```

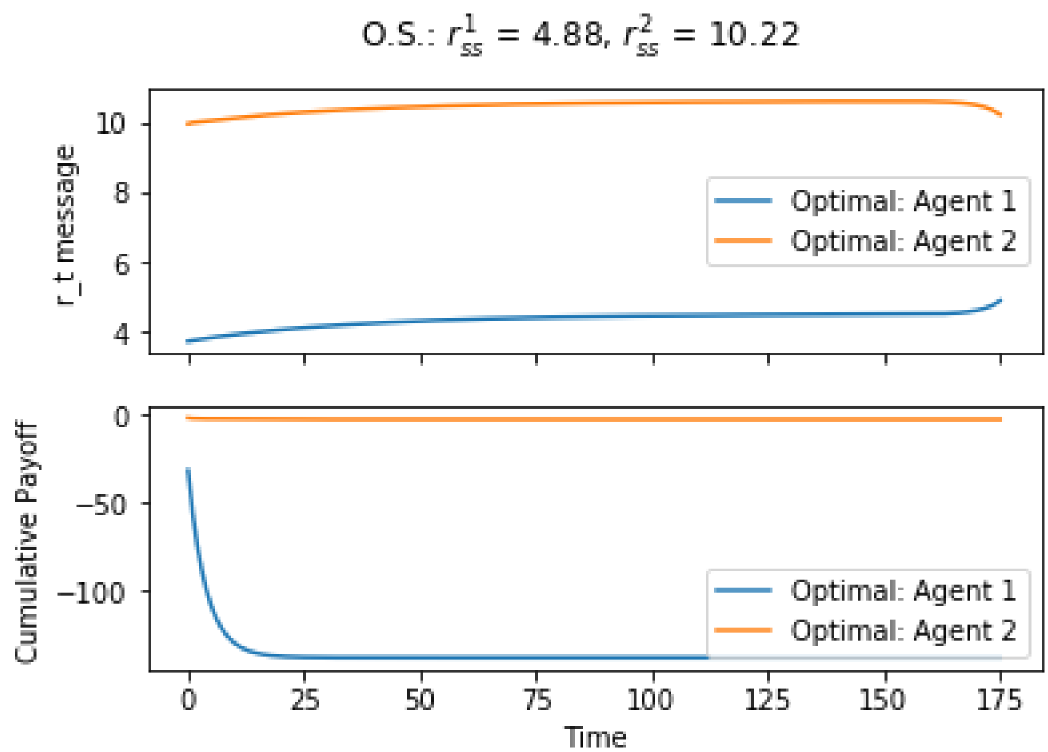
2.1 Experiment 3: Vary Initial Opinions (to verify Agent 2's payoff)

[34]:
```
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)

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

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_2]

X_0_b = np.array([
  10.1,
  8.72,
  12.5,
  11.21,
  9.73,
], ndmin = 2).T

X_0 = [X_0_b - (5 * np.ones((5, 1))), X_0_b - (10 * np.ones((5, 1)))]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [5, 10]
r = [5, 10]
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)]

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)
```



[35]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs)
```



This confirms that an agent with an agenda further away from the initial opinions of the naive agents will have far worse payoff.

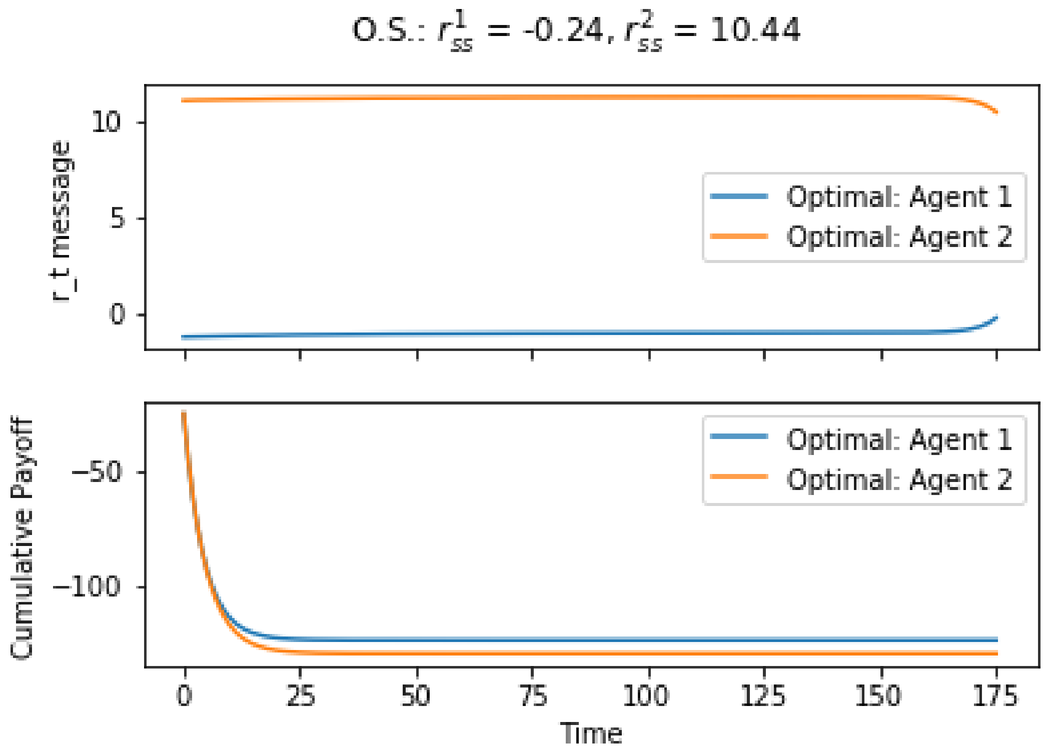
[36]:
```
xs[0][-1] + (5 * np.ones((5, 1)))
```

[36]:
```
array([[6.88493529],
       [7.31874643],
       [7.03353567],
       [7.01484687],
       [7.02619844]])
```

2.2 Experiment 4: Midpoint Hypothesis: Are the final opinions truly a biased average of the final steady state messages?

[37]:
```
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)

B_1 = np.array([
```

```
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_2]

X_0_b = np.array([
  5,
  5,
  5,
  5,
  5,
], ndmin = 2).T

X_0 = [X_0_b - (0 * np.ones((5, 1))), X_0_b - (10 * np.ones((5, 1)))]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 10]
r = [0, 10]
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)]

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)
```



[38]:
```
xs[0][-1]
```

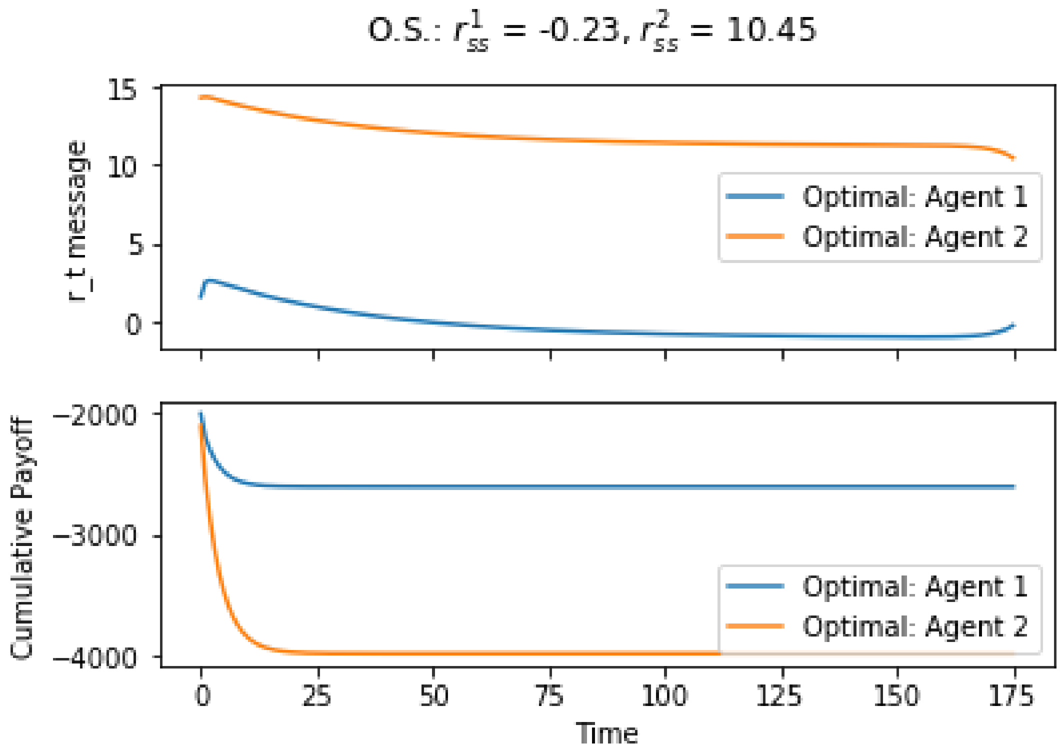
[38]:
```
array([[3.72651362],
       [4.59427596],
       [4.01843772],
       [3.98128532],
       [4.00355477]])
```

The answer is that there is a bias factor other than the state of the initial opinions, so no this is not a perfect midpoint.

2.3 Experiment 5: How strong is the influence of the strategic agents? Test with extreme opinions:

[39]:
```
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)

B_1 = np.array([
```

```
  0.0791,
  0,
  0,
  0,
  0,
], ndmin = 2).T

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_2]

X_0_b = np.array([
  50,
  50,
  0,
  -50,
  -50,
], ndmin = 2).T

X_0 = [X_0_b - (0 * np.ones((5, 1))), X_0_b - (10 * np.ones((5, 1)))]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 10]
r = [0, 10]
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)]

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)
```



Key finding: r_{ss} is still around the same value, but clearly the trajectories are quite different from before.

[40]:
```
xs[0][-1]
```

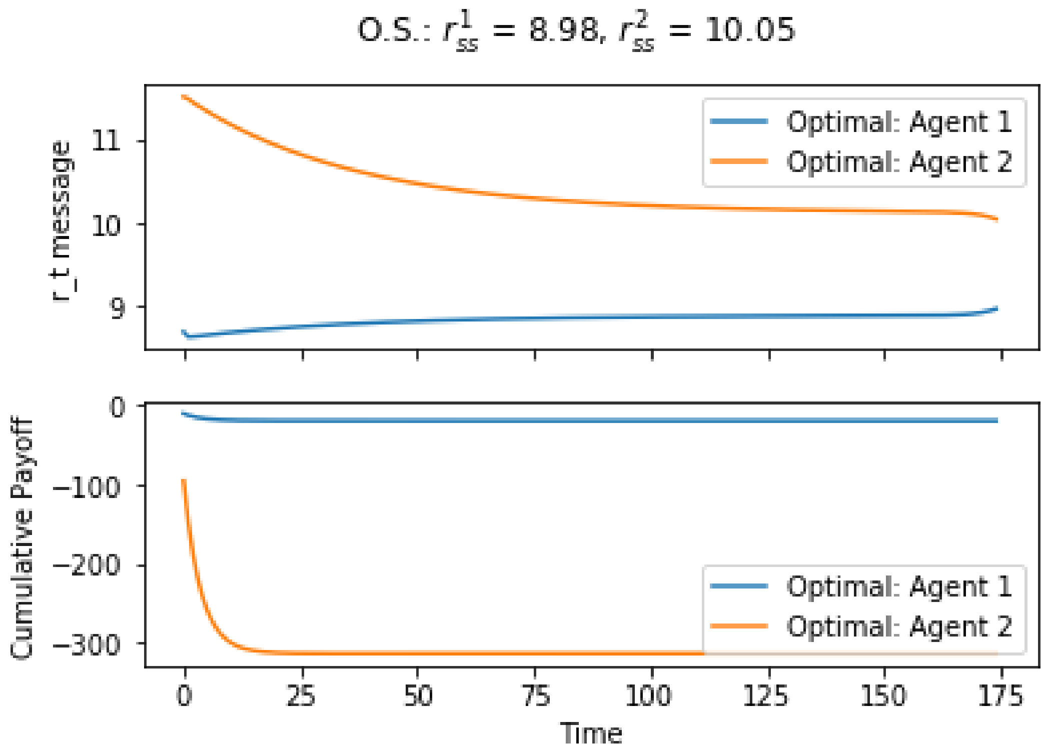
[40]:
```
array([[3.59046876],
       [4.45867056],
       [3.86583585],
       [3.8293901 ],
       [3.85029869]])
```

While the opinions are not perfectly the same as before, they are similar. This is indicative of a limit matrix similar to the ones seen before where only the strategic agents have influence in the long run. The question becomes what is this share of influence?

2.4 Experiment 6: Verification of Strategic Substitutes Theory

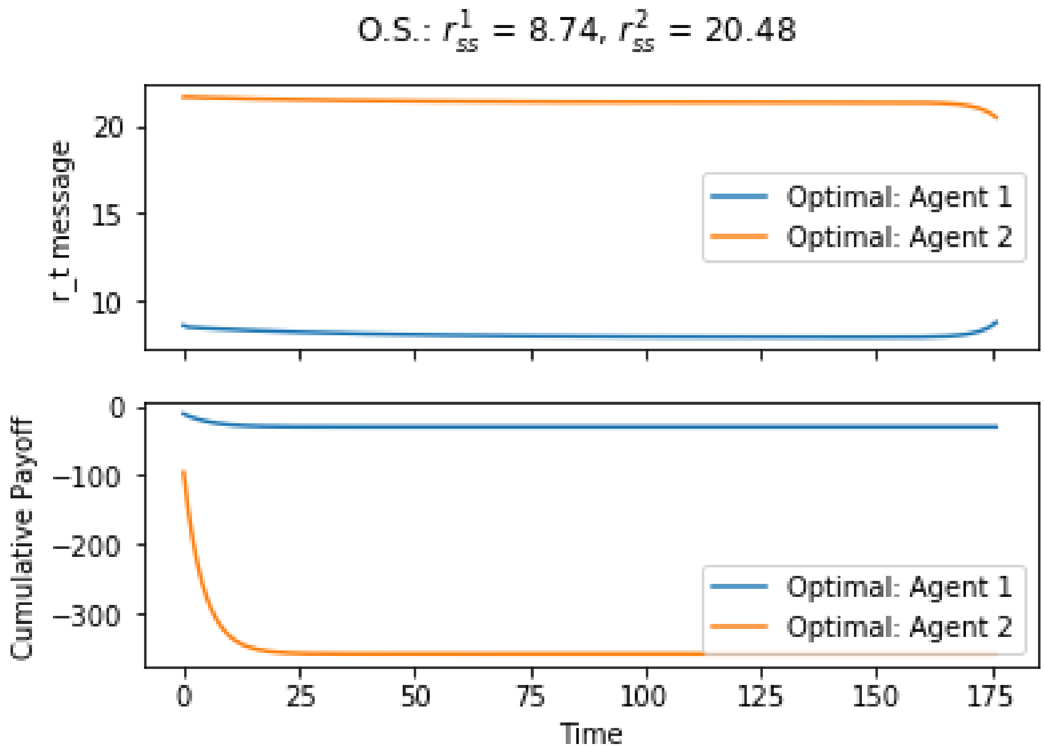
[41]:
```
A = np.array([
    [0.217,     0.2022,    0.2358,    0.1256,    0.1403],
    [0.8988*0.2497,    0.8988*0.0107,    0.8988*0.2334,    0.8988*0.1282,    0.8988*0.
↪378],
    [0.1285,    0.0907,    0.3185,    0.2507,    0.2116],
    [0.1975,    0.0629,    0.2863,    0.2396,    0.2137],
    [0.1256,    0.0711,    0.0253,    0.2244,    0.5536],
], ndmin = 2)
```

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

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_1, B_2]

X_0_b = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T

X_0 = [X_0_b - (0 * np.ones((5, 1))), X_0_b - (10 * np.ones((5, 1)))]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [9, 10]
r = [9, 10]
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)]

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)
```



[42]:
```
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)

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

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
```

```
], ndmin = 2).T

B = [B_1, B_2]

X_0_b = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T

X_0 = [X_0_b - (0 * np.ones((5, 1))), X_0_b - (10 * np.ones((5, 1)))]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [9, 20]
r = [9, 20]
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)]

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)
```

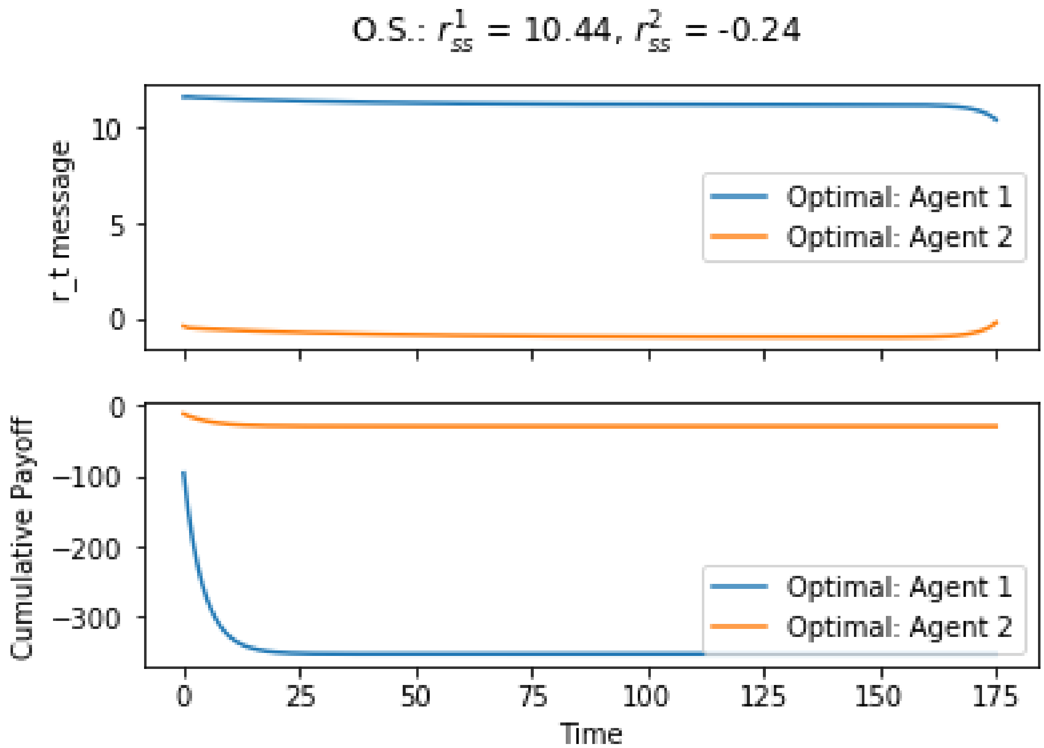


This appears to be the case. Notice the original steady states were -0.24, 10.44 with agendas of 0 and 10 respectively. When adjusting the agenda of agent 1 to 9, agent 2's steady state falls to 10.05 as agent 1's goes up. When then adjusting the agenda of agent 2 to 20, agent 1's steady state falls from 8.98 to 8.74 as agent 2's goes up.

2.5 Experiment 7: Double-check that the agents are swappable

[43]:
```
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)

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

```python
B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], ndmin = 2).T

B = [B_2, B_1]

X_0_b = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
], ndmin = 2).T

X_0 = [X_0_b - (10 * np.ones((5, 1))), X_0_b - (0 * np.ones((5, 1)))]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [10, 0]
r = [10, 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)]

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)
```



Yes, the agents are swappable.

2.6 Experiment 8: What is the limit matrix?

[44]:
```
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)

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

B_2 = np.array([
    0,
```

```
    0.1012,
    0,
    0,
    0,
], 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_2 = np.array([
    -0.98 - 10,
    -4.62 - 10,
    2.74  - 10,
    4.67  - 10,
    2.15  - 10,
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 10]
r = [0, 10]
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)]

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)
xs[0][-1]
```

[44]:
```
array([[3.70265674],
       [4.57049613],
       [3.99167739],
       [3.95464891],
       [3.97667971]])
```

[45]:
```
X_K = X_0_1
for i in range(len(rs[0])):
    X_K = A @ X_K + B_1 @ rs[0][i] + B_2 @ (rs[1][i] + 10 * np.ones((1, 1)))

print(X_K)
```

```
[[3.70265674]
 [4.57049613]
 [3.99167739]
 [3.95464891]
 [3.97667971]]
```

This method works. Merging the matrices:

[46]:
```
A = np.array([
  [0.217,           0.2022,          0.2358,          0.1256,          0.1403, ␣
 ↪     0, 0.0791],
  [0.8988*0.2497,   0.8988*0.0107,   0.8988*0.2334,   0.8988*0.1282,   0.8988*0.
 ↪378, 0.1012, 0],
  [0.1285,    0.0907,    0.3185,    0.2507,    0.2116, 0, 0],
  [0.1975,    0.0629,    0.2863,    0.2396,    0.2137, 0, 0],
  [0.1256,    0.0711,    0.0253,    0.2244,    0.5536, 0, 0],
  [0, 0, 0, 0, 0, 1, 0],
  [0, 0, 0, 0, 0, 0, 1]
], ndmin = 2)
val = np.linalg.matrix_power(A, 1000000000000)
np.round(val, 5)
```

[46]:
```
array([[0.      , 0.      , 0.      , 0.      , 0.      , 0.38171, 0.61829],
       [0.      , 0.      , 0.      , 0.      , 0.      , 0.46321, 0.53679],
       [0.      , 0.      , 0.      , 0.      , 0.      , 0.41026, 0.58974],
       [0.      , 0.      , 0.      , 0.      , 0.      , 0.40686, 0.59314],
       [0.      , 0.      , 0.      , 0.      , 0.      , 0.40895, 0.59105],
       [0.      , 0.      , 0.      , 0.      , 0.      , 1.      , 0.      ],
       [0.      , 0.      , 0.      , 0.      , 0.      , 0.      , 1.      ]])
```

So stagewise, we get:

[47]:
```
X_K = np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
  0,
  0,
], ndmin = 2).T
for i in range(len(rs[0])):
    X_K[5] = rs[1][i].item() + 10
    X_K[6] = rs[0][i].item()
```

```
    X_K = A @ X_K

print(X_K)
```

```
[[ 3.70265674]
 [ 4.57049613]
 [ 3.99167739]
 [ 3.95464891]
 [ 3.97667971]
 [10.44076351]
 [-0.23791536]]
```

which is the same result.

[48]:
```
target = val @ np.array([
    -0.98,
    -4.62,
    2.74,
    4.67,
    2.15,
    rs[1][-1].item() + 10,
    rs[0][-1].item(),
], ndmin = 2).T

target
```

[48]:
```
array([[ 3.83822212],
       [ 4.70851624],
       [ 4.14316306],
       [ 4.10677968],
       [ 4.12912094],
       [10.44076351],
       [-0.23791536]])
```

Note the limit, however, is wrong. If the calculations are extended:

[49]:
```
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)

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

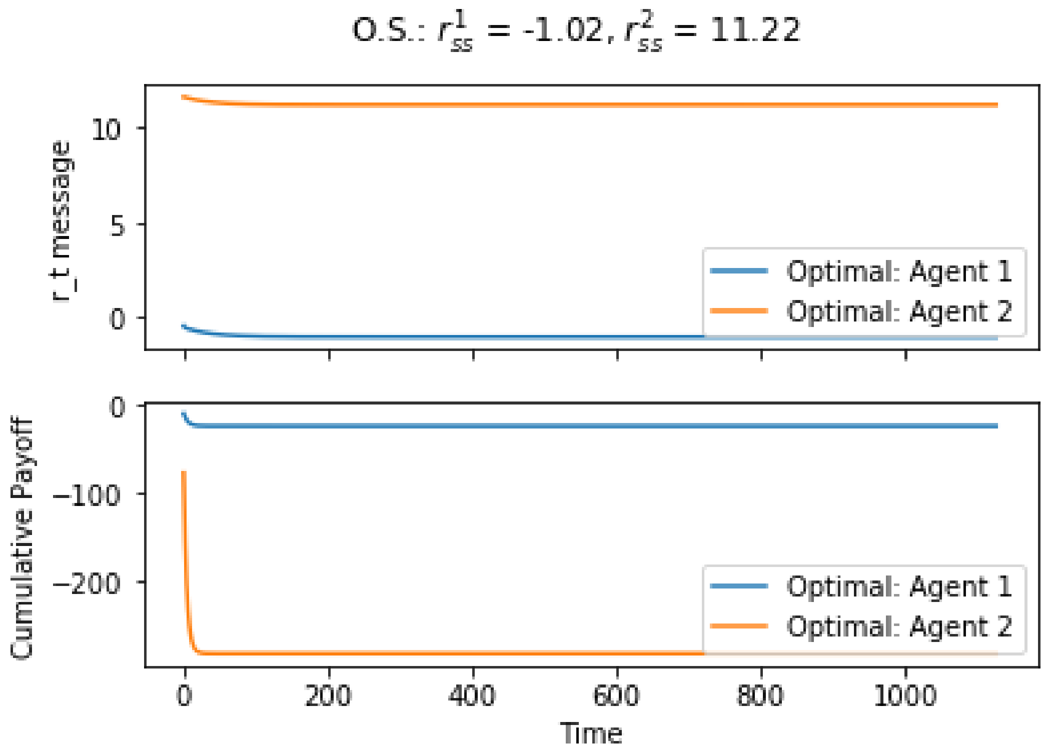
```
  0,
], ndmin = 2).T

B_2 = np.array([
  0,
  0.1012,
  0,
  0,
  0,
], 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_2 = np.array([
  -0.98 - 10,
  -4.62 - 10,
  2.74  - 10,
  4.67  - 10,
  2.15  - 10,
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.8
n = 5
m = 1
L = 2
Q = [0.2 * np.identity(n), 0.2 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 10]
r = [0, 10]
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)]

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)

X_t = [a.copy() for a in X_0]
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 True:
    i += 1
    K_t = historical_K[0]
    k_t = historical_k[0]
    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)
    for l in range(L):
        Y_new = rs[l][-1]
        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 l == 1 and abs(np.max(X_t[l] - X_new)) == 0:
            break
        X_t[l] = X_new
    else:
        continue
    break
```

[50]:
```
xs[0][-1]
```

[50]:
```
array([[3.65276565],
       [4.6496899 ],
       [4.00207631],
       [3.96039905],
       [3.98599103]])
```

[51]:
```
do_plot(rs, r, payoffs)
```



[52]:
```
print(len(payoffs[0]))
```

```
1126
```

[53]:
```
target_2 = val @ np.array([
  -0.98,
  -4.62,
  2.74,
  4.67,
  2.15,
  rs[1][-1].item() + 10,
  rs[0][-1].item(),
], ndmin = 2).T
target_2
```

[53]:
```
array([[ 3.65276565],
       [ 4.6496899 ],
       [ 4.00207631],
       [ 3.96039905],
       [ 3.98599103],
       [11.21599459],
       [-1.01646082]])
```

[54]:
```
target_2[:5] - xs[0][-1]
```

[54]:
```
array([[7.37188088e-14],
       [7.46069873e-14],
       [8.34887715e-14],
       [8.21565038e-14],
       [8.26005930e-14]])
```

Now they match.