Opposite_Bias_Finite

November 21, 2021

1 Finite-Horizon Version of Opposite Bias Model

James Yu, 21 November 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, with the correct pairings:
    base = [[(B[l_prime].T @ K[l_prime] @ B[l]).item() for l in range(L)] 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_finite(K_t, k_t, kappa_t, A, B, delta, n, m, L, Q, R, x, c, T):
    historical_K = [K_t]
    historical_k = [k_t]
    historical_kappa = [kappa_t]
    max_distances = defaultdict(list)
    counter = 0
    for i in range(T):
        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

    return max_distances, historical_K, historical_k, historical_kappa
```

```
[5]: 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)
    i = 0
    while i < 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)
            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
        i += 1
```

```
    return xs, rs, payoffs
```

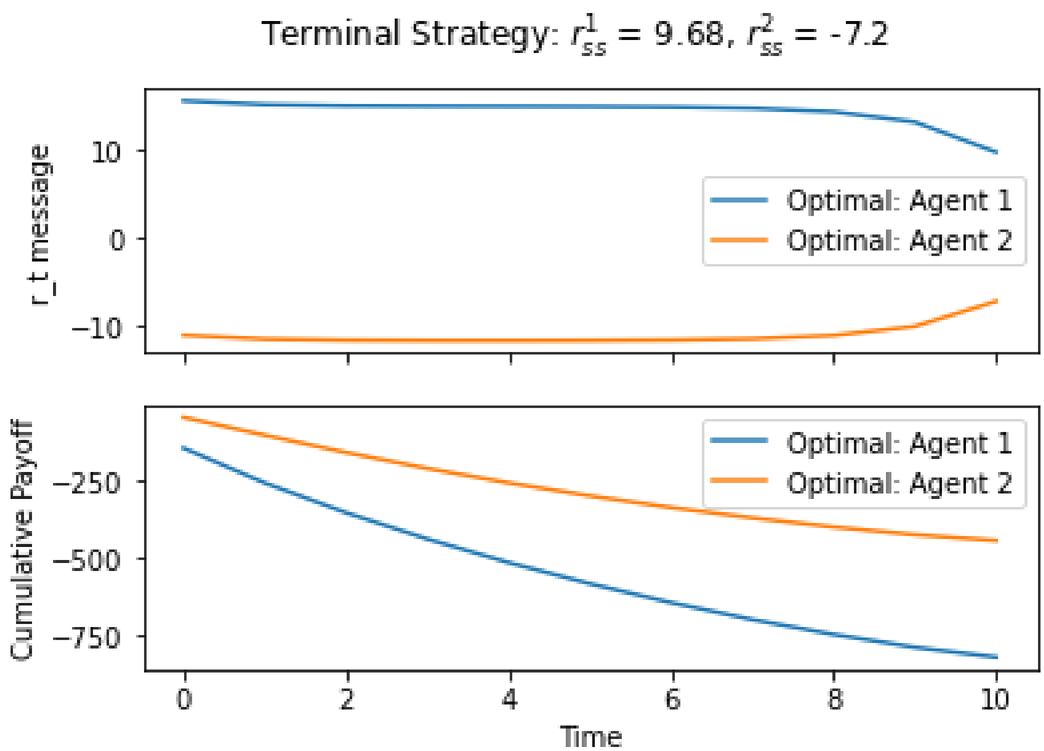
2 1. Baseline model, r__1 = r__2 = 0

[6]:
```
A = np.array([
  [0.5],
], ndmin = 2)

B_1 = np.array([
  0.25,
], ndmin = 2).T

B_2 = np.array([
  0.25,
], ndmin = 2).T

B = [B_1, B_2]

x0 = 0

X_0_1 = np.array([ # \chi_0^1
  x0 - 10, # agenda here is 10
], ndmin = 2).T

X_0_2 = np.array([ # \chi_0^2
  x0 + 5, # agenda here is -5
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.9
n = 1
m = 1
L = 2
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [10, -5]
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)]
```

[7]:
```
max_distances, historical_K, historical_k, historical_kappa = solve_finite(Q,␣
↪[np.zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x,␣
↪c, 10) # 10 periods
```

[8]:
```
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
```

[9]:
```
def do_plot(rs, r, payoffs, num_agents = 1, set_cap = np.inf, flag = False,␣
↪legend = True):
    fig, sub = plt.subplots(2, sharex=True)
    if legend:
        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(num_agents)])}")

    for l in range(num_agents):
        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',␣
↪'Channel'][flag]} {l+1}")
    sub[0].set(ylabel = "r_t message")

    for l in range(num_agents):
        sub[1].plot(range(min(len(payoffs[l]), set_cap)), payoffs[l][:
↪min(len(payoffs[l]), set_cap)], label = f"Optimal: {['Agent',␣
↪'Channel'][flag]} {l+1}")
    sub[1].set(xlabel = "Time", ylabel =  "Cumulative Payoff")
    if legend:
        sub[0].legend()
        sub[1].legend()
    plt.show()
do_plot(rs, r, payoffs, num_agents = 2, set_cap = 1000)
```



[10]:
```
print([round(a.item(), 4) for a in rs[0]]) # r^1
```

```
[15.4559, 15.0264, 14.8942, 14.8515, 14.8324, 14.81, 14.7571, 14.6132, 14.2118,
13.0726, 9.678]
```

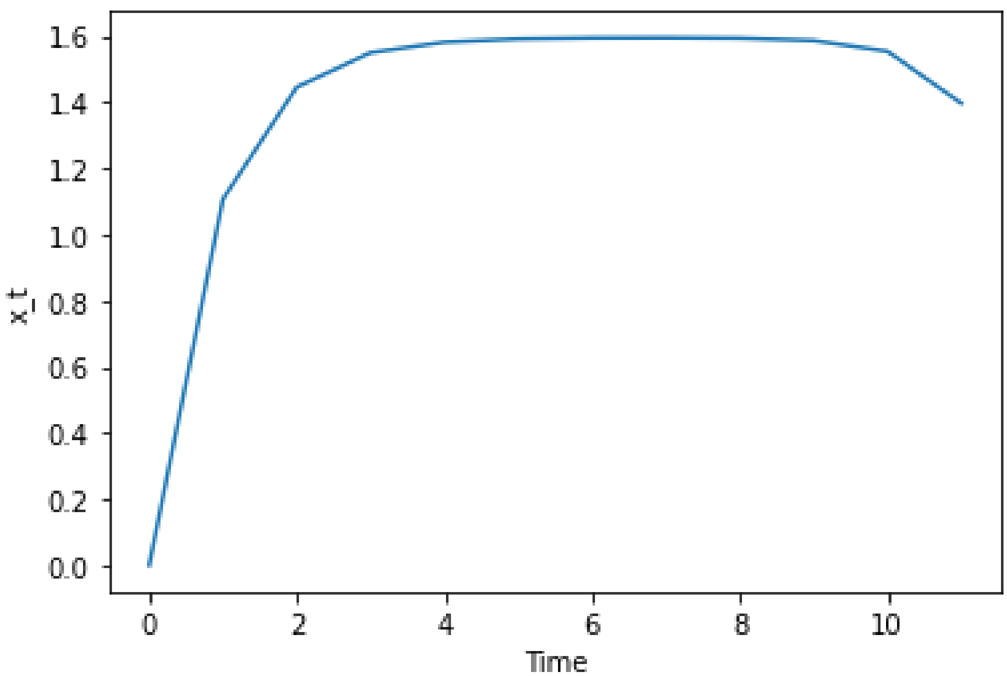
[11]:
```
print([round(a.item(), 4) for a in rs[1]]) # r^2
```

```
[-11.0268, -11.4556, -11.5861, -11.6239, -11.6296, -11.6149, -11.5657, -11.428,
-11.0521, -10.0293, -7.197]
```

[12]:
```
print([round(a.item() + 10, 4) for a in xs[0]]) # x_t
```

```
[0, 1.1073, 1.4464, 1.5502, 1.582, 1.5917, 1.5946, 1.5952, 1.5939, 1.5869,
1.5543, 1.3974]
```

[13]:
```
plt.plot(range(len(xs[0])), [a.item() + 10 for a in xs[0]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



[14]:
```
historical_K
```

[14]:
```
[[array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.12498245]]), array([[1.12498245]])],
 [array([[1.12498154]]), array([[1.12498154]])],
 [array([[1.12496399]]), array([[1.12496399]])],
 [array([[1.12462445]]), array([[1.12462445]])],
 [array([[1.11808]]), array([[1.11808]])],
 [array([[1.]]), array([[1.]])]]
```

[15]:
```
historical_k
```

[15]:
```
[[array([[-7.46885269]]), array([[5.86199258]])],
 [array([[-7.46828106]]), array([[5.86142299]])],
 [array([[-7.46670367]]), array([[5.85985479]])],
 [array([[-7.46234786]]), array([[5.85554031]])],
 [array([[-7.45030571]]), array([[5.84368421]])],
 [array([[-7.41695172]]), array([[5.81116743]])],
 [array([[-7.32429624]]), array([[5.72227842]])],
 [array([[-7.06580784]]), array([[5.48071365]])],
 [array([[-6.34221115]]), array([[4.83273583]])],
 [array([[-4.3542]]), array([[3.1734]])],
 [array([[0.]]), array([[0.]])]]
```

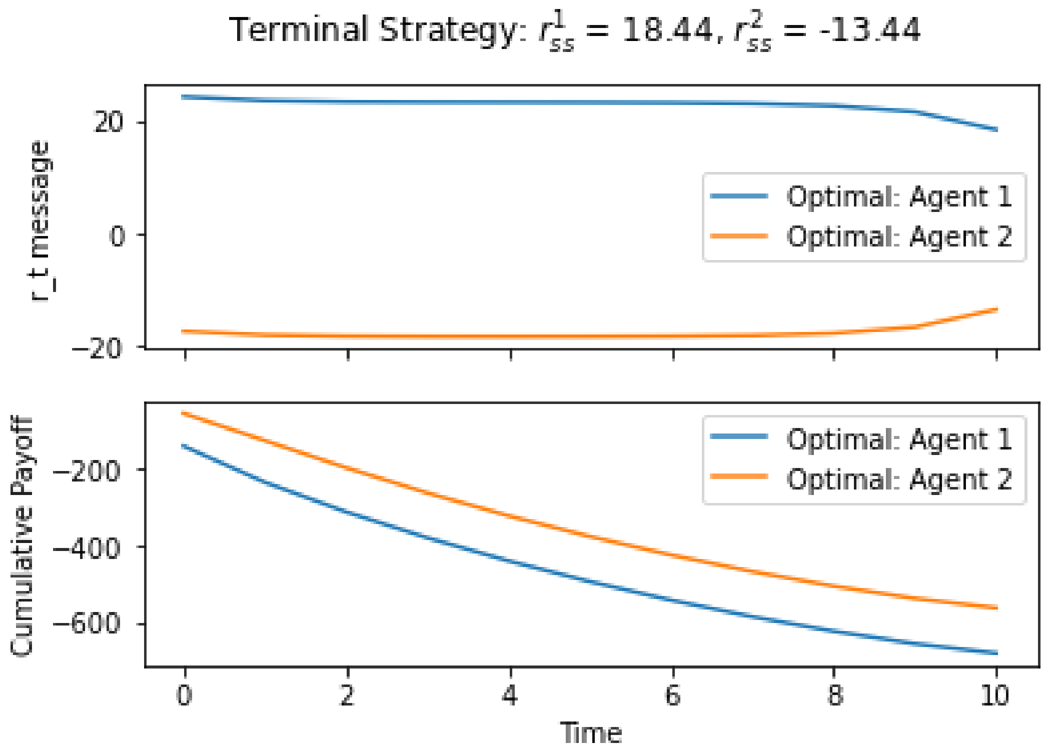
[16]:
```
historical_kappa
```

[16]:
```
[[array([[-613.7756583]]), array([[-374.71689512]])],
 [array([[-570.38636859]]), array([[-347.61413298]])],
 [array([[-522.18978856]]), array([[-317.51070904]])],
 [array([[-468.67594092]]), array([[-284.09208067]])],
 [array([[-409.32073016]]), array([[-247.04178682]])],
 [array([[-343.65941944]]), array([[-206.09885514]])],
 [array([[-271.50095776]]), array([[-161.22115347]])],
 [array([[-193.53325401]]), array([[-113.03375262]])],
 [array([[-113.0006811]]), array([[-64.01402496]])],
 [array([[-40.14028125]]), array([[-21.32128125]])],
 [0, 0]]
```

3 2. $r_1 = x_1$, $r_2 = x_2$

[17]:
```
A = np.array([
    [0.5],
], ndmin = 2)
```

```
B_1 = np.array([
  0.25,
], ndmin = 2).T

B_2 = np.array([
  0.25,
], ndmin = 2).T

B = [B_1, B_2]

x0 = 0

X_0_1 = np.array([ # \chi_0^1
  x0 - 10, # agenda here is 10
], ndmin = 2).T

X_0_2 = np.array([ # \chi_0^2
  x0 + 5, # agenda here is -5
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.9
n = 1
m = 1
L = 2
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [10, -5]
r = [10, -5] # now add cost dependent on the agenda
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_finite(Q,␣
 ↪[np.zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x,␣
 ↪c, 10)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, num_agents = 2, set_cap = 1000)
```



[18]:
```
print([round(a.item(), 4) for a in rs[0]]) # r^1
```

```
[14.2102, 13.5377, 13.331, 13.2655, 13.2395, 13.2151, 13.1622, 13.0208, 12.632,
11.551, 8.4375]
```

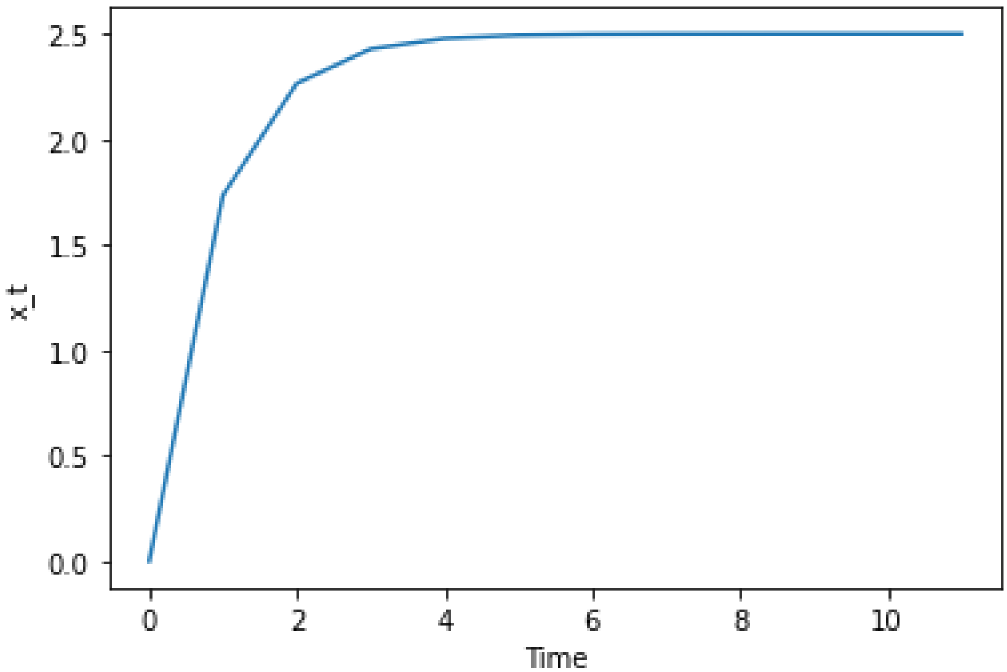
[19]:
```
print([round(a.item(), 4) for a in rs[1]]) # r^2
```

```
[-12.2724, -12.9443, -13.1493, -13.2099, -13.2225, -13.2098, -13.1606, -13.0203,
-12.6319, -11.5509, -8.4375]
```

[20]:
```
print([round(a.item() + 10, 4) for a in xs[0]]) # x_t
```

```
[0, 1.7344, 2.2656, 2.4282, 2.478, 2.4933, 2.4979, 2.4994, 2.4998, 2.4999, 2.5,
2.5]
```

[21]:
```
plt.plot(range(len(xs[0])), [a.item() + 10 for a in xs[0]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



[22]:
```
historical_K
```

[22]:
```
[[array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.12498245]]), array([[1.12498245]])],
 [array([[1.12498154]]), array([[1.12498154]])],
 [array([[1.12496399]]), array([[1.12496399]])],
 [array([[1.12462445]]), array([[1.12462445]])],
 [array([[1.11808]]), array([[1.11808]])],
 [array([[1.]]), array([[1.]])]]
```

[23]:
```
historical_k
```

[23]:
```
[[array([[-6.66542263]]), array([[6.66542263]])],
 [array([[-6.66485202]]), array([[6.66485202]])],
 [array([[-6.66327923]]), array([[6.66327923]])],
 [array([[-6.65894408]]), array([[6.65894408]])],
 [array([[-6.64699496]]), array([[6.64699496]])],
 [array([[-6.61405958]]), array([[6.61405958]])],
 [array([[-6.52328733]]), array([[6.52328733]])],
 [array([[-6.27326075]]), array([[6.27326075]])],
 [array([[-5.58747349]]), array([[5.58747349]])],
```

```
 [array([[-3.7638]]), array([[3.7638]])],
 [array([[0.]]), array([[0.]])]]
```

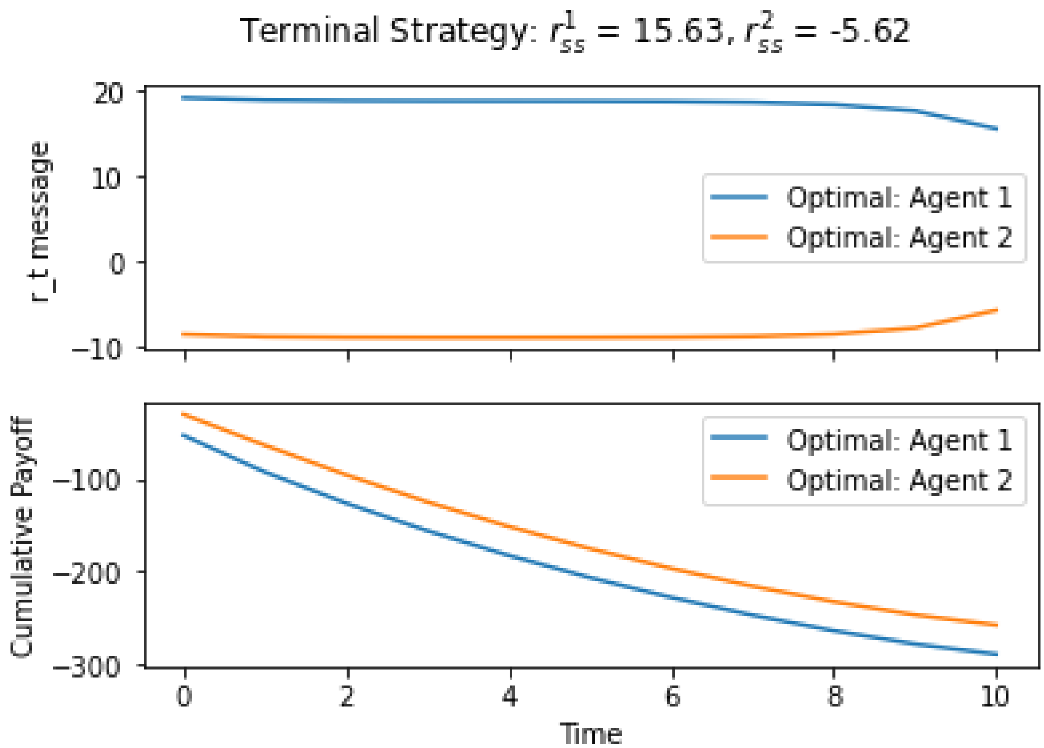
[24]:
```
historical_kappa
```

[24]:
```
[[array([[-486.89255588]]), array([[-486.89255588]])],
 [array([[-452.12064856]]), array([[-452.12064856]])],
 [array([[-413.49744303]]), array([[-413.49744303]])],
 [array([[-370.6165254]]), array([[-370.6165254]])],
 [array([[-323.06408846]]), array([[-323.06408846]])],
 [array([[-270.48435717]]), array([[-270.48435717]])],
 [array([[-212.76810826]]), array([[-212.76810826]])],
 [array([[-150.57777277]]), array([[-150.57777277]])],
 [array([[-86.77060797]]), array([[-86.77060797]])],
 [array([[-29.99278125]]), array([[-29.99278125]])],
 [0, 0]]
```

4 3. set $x_0 = 4$ and $x_2 = 0$

[25]:
```
A = np.array([
  [0.5],
], ndmin = 2)

B_1 = np.array([
  0.25,
], ndmin = 2).T

B_2 = np.array([
  0.25,
], ndmin = 2).T

B = [B_1, B_2]

x0 = 4

X_0_1 = np.array([ # \chi_0^1
  x0 - 10, # agenda here is 10
], ndmin = 2).T

X_0_2 = np.array([ # \chi_0^2
  x0 + 0, # agenda here is 0
], ndmin = 2).T
X_0 = [X_0_1, X_0_2]

delta = 0.9
n = 1
```

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

x = [10, 0]
r = [10, 0] # now add cost dependent on the agenda
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_finite(Q,␣
↪[np.zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x,␣
↪c, 10)
xs, rs, payoffs = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs, r, payoffs, num_agents = 2, set_cap = 1000)
```



[26]:
```
print([round(a.item(), 4) for a in rs[0]]) # r^1
```

```
[9.2151, 8.946, 8.8631, 8.8363, 8.8241, 8.8093, 8.7746, 8.6805, 8.4213, 7.7006,
5.625]
```

[27]:
```
print([round(a.item(), 4) for a in rs[1]]) # r^2
```

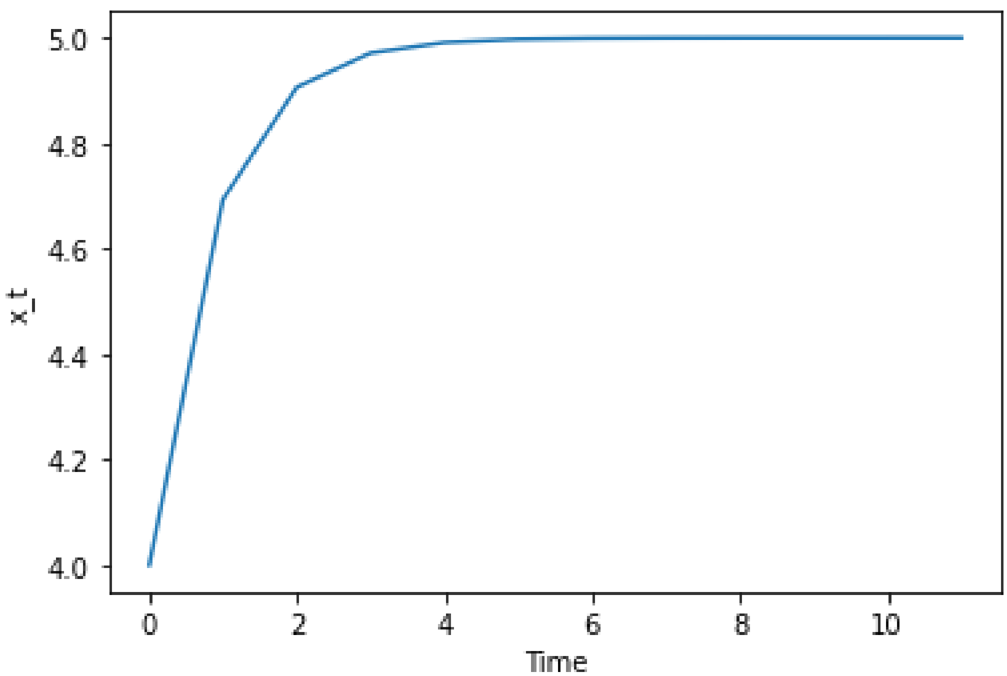
```
[-8.44, -8.7087, -8.7904, -8.814, -8.8172, -8.8073, -8.7739, -8.6803, -8.4213,
```

```
-7.7006, -5.625]
```

[28]:
```
print([round(a.item() + 10, 4) for a in xs[0]]) # x_t
```

```
[4, 4.6938, 4.9062, 4.9713, 4.9912, 4.9973, 4.9992, 4.9997, 4.9999, 5.0, 5.0,
5.0]
```

[29]:
```
plt.plot(range(len(xs[0])), [a.item() + 10 for a in xs[0]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



[30]:
```
historical_K
```

[30]:
```
[[array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.1249825]]), array([[1.1249825]])],
 [array([[1.12498245]]), array([[1.12498245]])],
 [array([[1.12498154]]), array([[1.12498154]])],
 [array([[1.12496399]]), array([[1.12496399]])],
 [array([[1.12462445]]), array([[1.12462445]])],
 [array([[1.11808]]), array([[1.11808]])],
 [array([[1.]]), array([[1.]])]]
```

```
[31]: historical_k
```

```
[31]: [[array([[-4.44361509]]), array([[4.44361509]])],
       [array([[-4.44323468]]), array([[4.44323468]])],
       [array([[-4.44218616]]), array([[4.44218616]])],
       [array([[-4.43929606]]), array([[4.43929606]])],
       [array([[-4.43132998]]), array([[4.43132998]])],
       [array([[-4.40937305]]), array([[4.40937305]])],
       [array([[-4.34885822]]), array([[4.34885822]])],
       [array([[-4.18217383]]), array([[4.18217383]])],
       [array([[-3.72498233]]), array([[3.72498233]])],
       [array([[-2.5092]]), array([[2.5092]])],
       [array([[0.]]), array([[0.]])]]
```

```
[32]: historical_kappa
```

```
[32]: [[array([[-216.3966915]]), array([[-216.3966915]])],
       [array([[-200.94251047]]), array([[-200.94251047]])],
       [array([[-183.77664135]]), array([[-183.77664135]])],
       [array([[-164.71845573]]), array([[-164.71845573]])],
       [array([[-143.58403932]]), array([[-143.58403932]])],
       [array([[-120.21526985]]), array([[-120.21526985]])],
       [array([[-94.56360367]]), array([[-94.56360367]])],
       [array([[-66.92345457]]), array([[-66.92345457]])],
       [array([[-38.56471465]]), array([[-38.56471465]])],
       [array([[-13.330125]]), array([[-13.330125]])],
       [0, 0]]
```