Opposite_Bias_New

November 21, 2021

1 Opposite Bias Model Further Questions

James Yu, 21 November 2021

1.1 Using $R = 0.2I$ for consistency.

[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_

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 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)
```

```python
            if l == L - 1 and 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
```

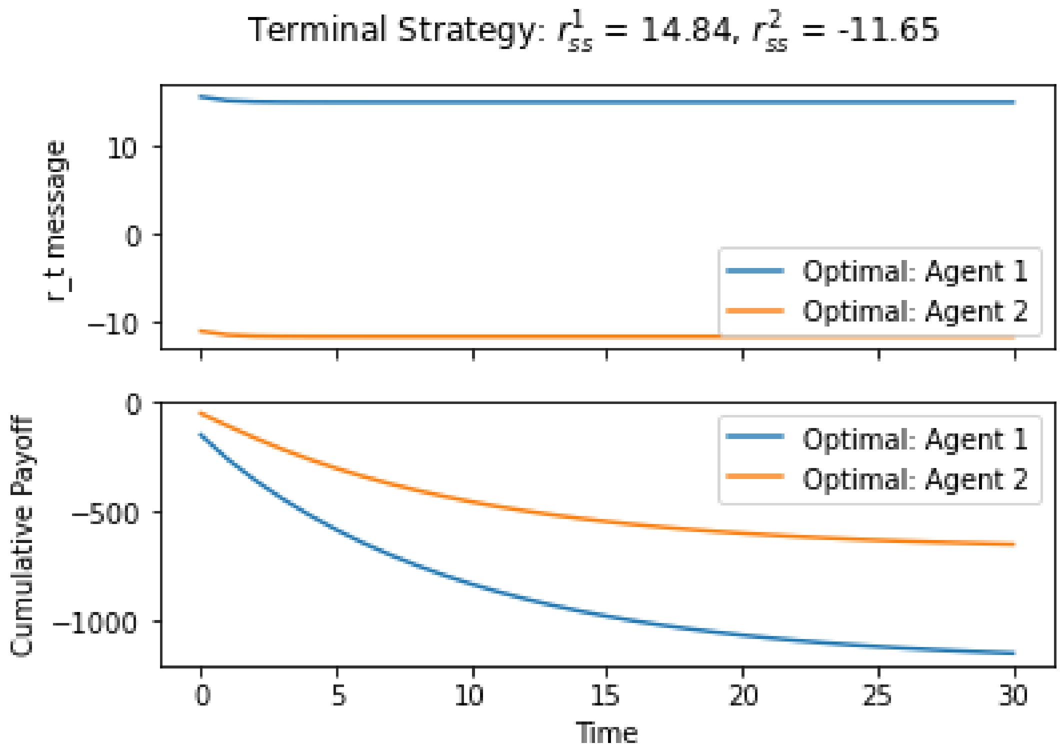
2 1. What do r_t^1, r_2^2 and x_t look like in $r_1 = r_2 = 0$?

[6]:
```python
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(Q, [np.
↪zeros((1, n)), np.zeros((1, n))], [0, 0], A, B, delta, n, m, L, Q, R, x, c,␣
↪tol = 1000)
```

[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)
```



2.1 r_t^1:

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

```
[15.4561, 15.027, 14.8955, 14.8553, 14.843, 14.8392, 14.838, 14.8377, 14.8376,
14.8376, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375,
14.8375, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375, 14.8375,
14.8375, 14.8375, 14.8375, 14.8375]
```

2.2 r_t^2:

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

```
[-11.027, -11.4561, -11.5875, -11.6277, -11.6401, -11.6438, -11.645, -11.6454,
-11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455,
-11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455,
-11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455, -11.6455]
```

```
[15]: print([round(a.item() - b.item(), 4) for a, b in zip(rs[0], rs[1])])
```

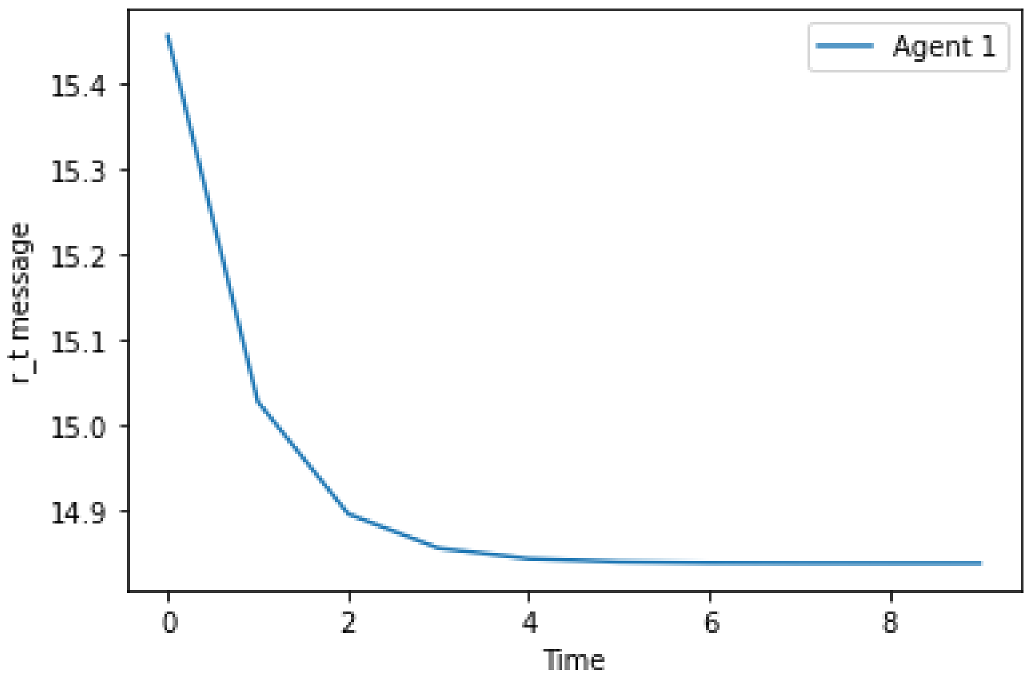
```
[26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483,
26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483,
```

```
26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483, 26.483,
26.483]
```

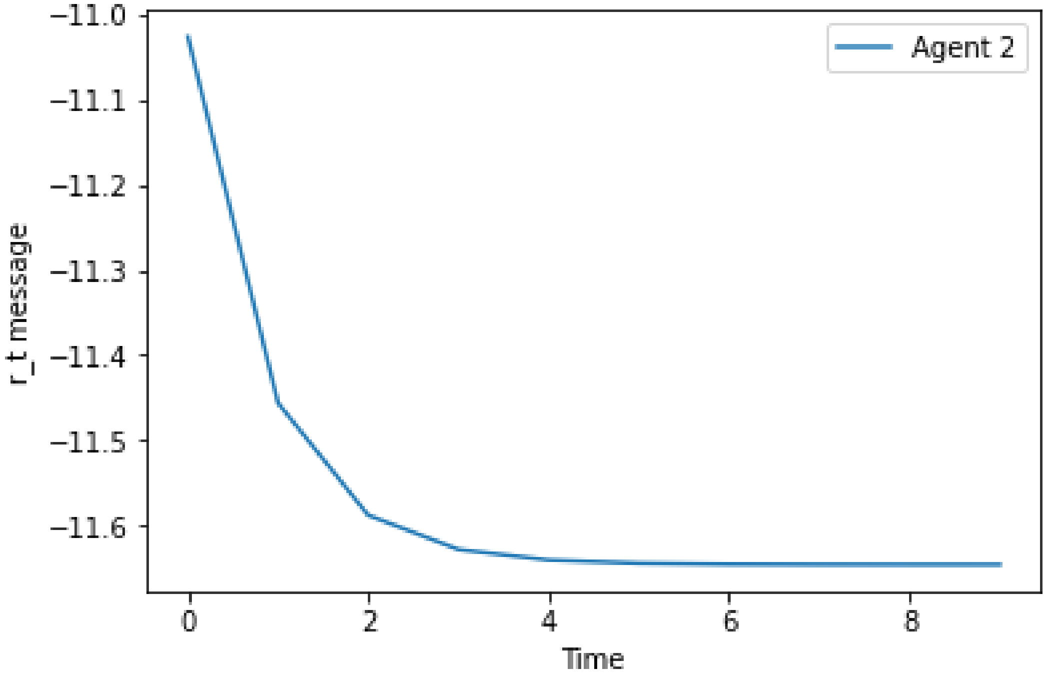
The two strategies are in fact entirely equidistant from each other.

If we take the early periods to be before convergence, this is about the first ten periods:

[12]:
```
num = 10
plt.plot(range(num), [a.item() for a in rs[0][:num]], label = "Agent 1")
plt.xlabel("Time")
plt.ylabel("r_t message")
plt.legend()
plt.show()
```



[13]:
```
plt.plot(range(num), [a.item() for a in rs[1][:num]], label = "Agent 2")
plt.xlabel("Time")
plt.ylabel("r_t message")
plt.legend()
plt.show()
```

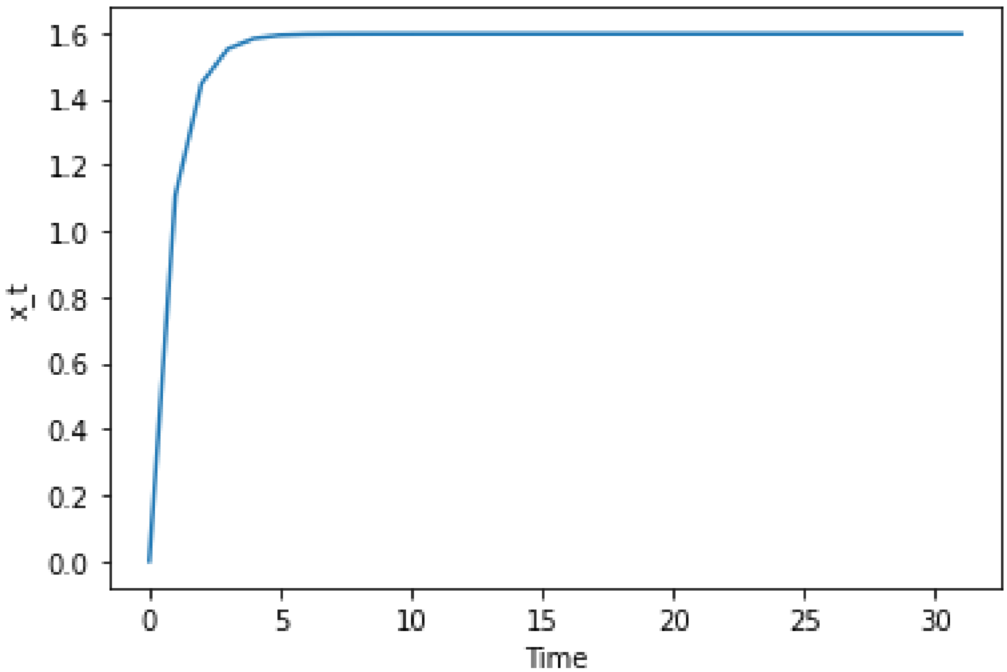


2.3 x_t:

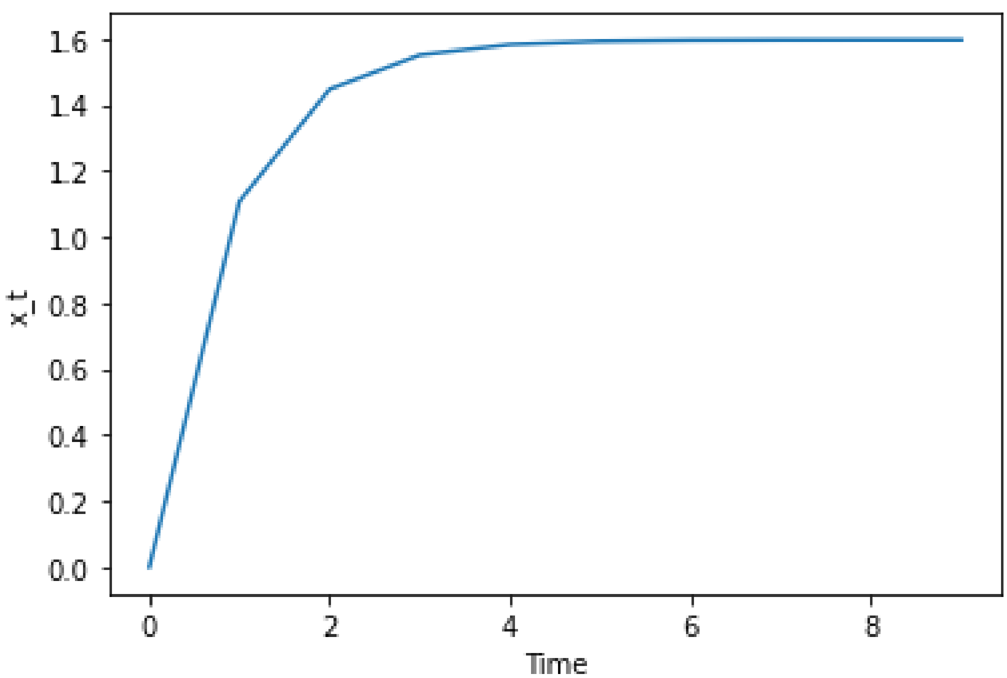
[14]:
```
print([round(a.item() + 10, 4) for a in xs[0]]) # + 10 since xs is the \chi␣
↪variable
```

```
[0, 1.1073, 1.4464, 1.5502, 1.582, 1.5917, 1.5947, 1.5956, 1.5959, 1.596, 1.596,
1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596,
1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596, 1.596]
```

[15]:
```
plt.plot(range(len(xs[0])), [a.item() + 10 for a in xs[0]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



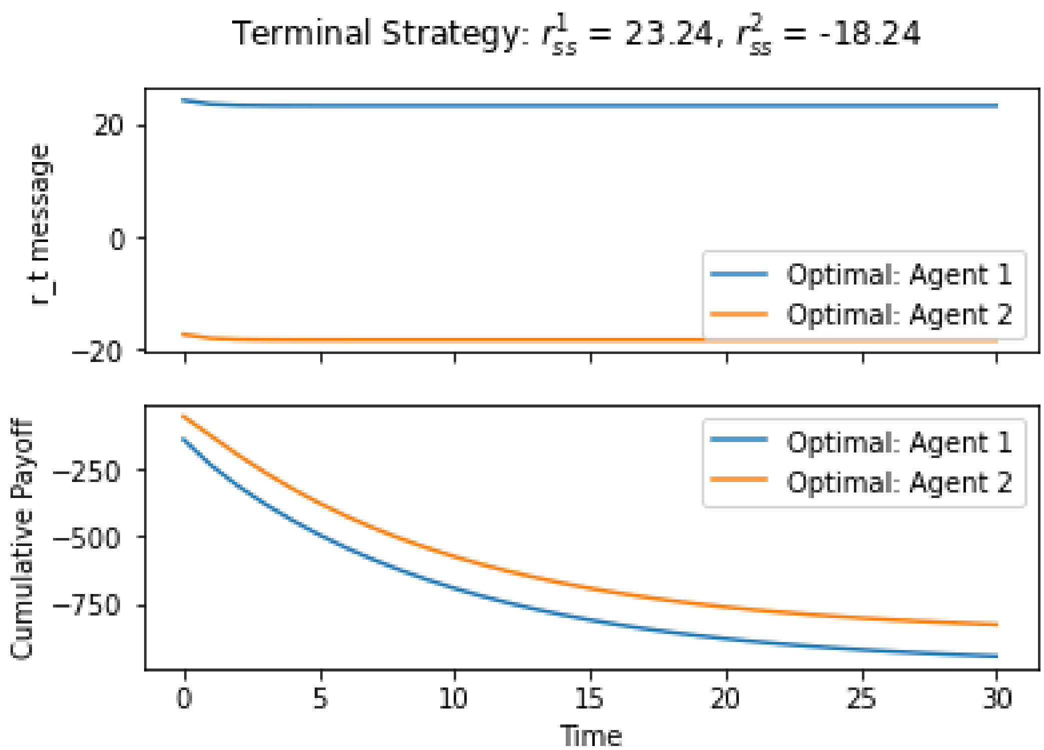
[16]:
```
plt.plot(range(10), [a.item() + 10 for a in xs[0][:10]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



3 2. What happens when we add agenda-dependent cost to the model?

[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(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, num_agents = 2, set_cap = 1000)
```



3.1 r_t^1:

[18]:
```
print([round(a.item() + 10, 4) for a in rs[0]])
```

[24.2104, 23.5382, 23.3324, 23.2693, 23.25, 23.2441, 23.2423, 23.2418, 23.2416, 23.2415]

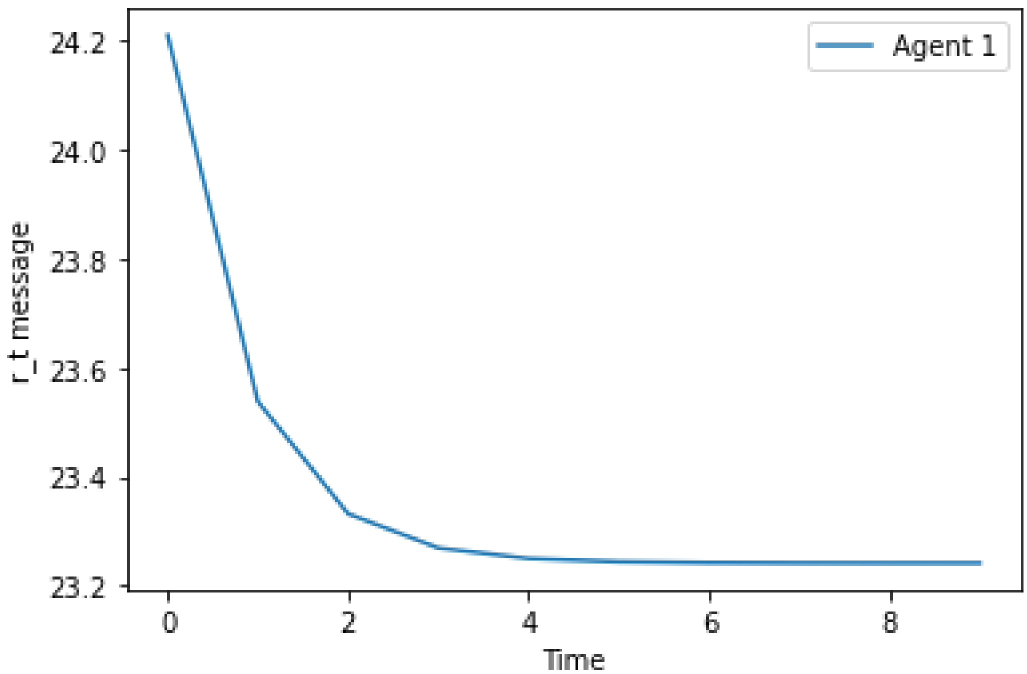
3.2 r_t^2:

[19]:
```
print([round(a.item() - 5, 4) for a in rs[1]])
```

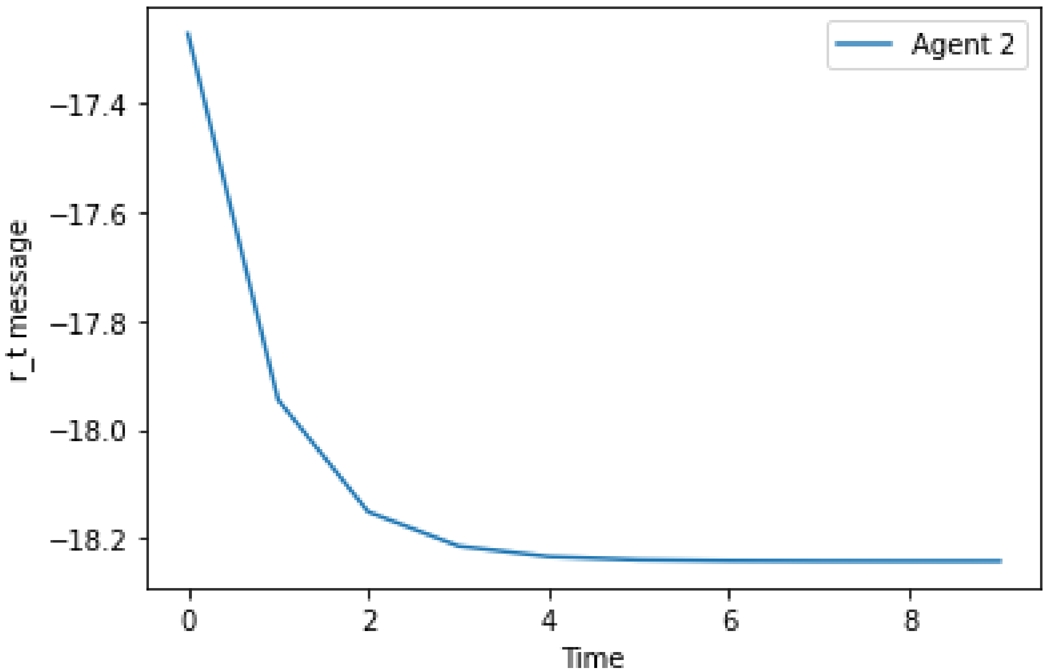
[-17.2726, -17.9448, -18.1507, -18.2137, -18.233, -18.2389, -18.2407, -18.2413, -18.2414, -18.2415]

[20]:
```
num = 10
plt.plot(range(num), [a.item() + 10 for a in rs[0][:num]], label = "Agent 1")
```

```
plt.xlabel("Time")
plt.ylabel("r_t message")
plt.legend()
plt.show()
```



[21]:
```
plt.plot(range(num), [a.item() - 5 for a in rs[1][:num]], label = "Agent 2")
plt.xlabel("Time")
plt.ylabel("r_t message")
plt.legend()
plt.show()
```



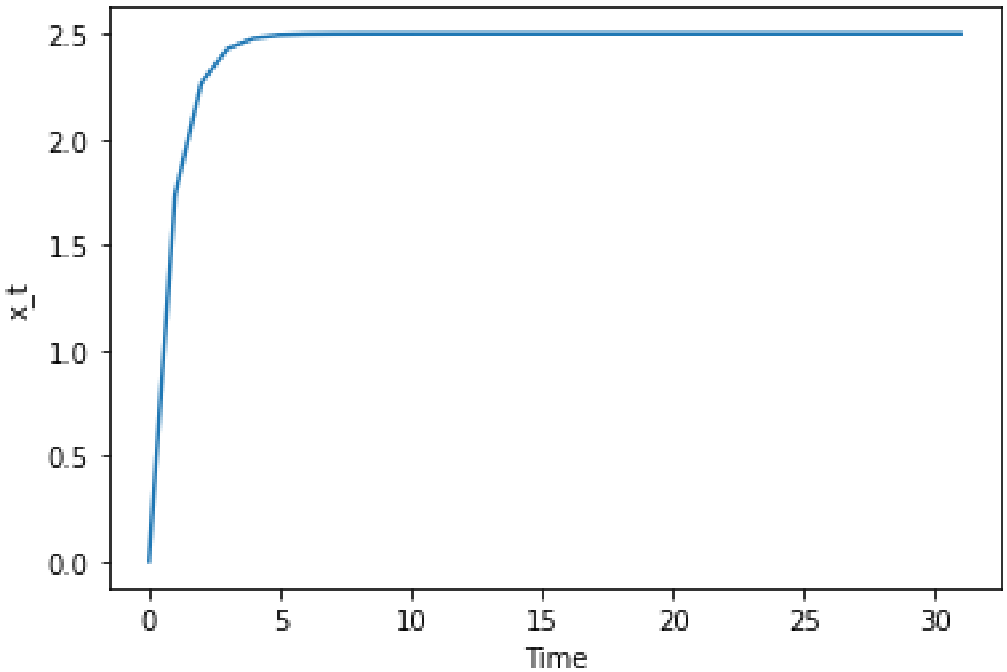
3.3 x_t:

[22]:
```
print([round(a.item() + 10, 4) for a in xs[0]]) # + 10 since xs is the \chi␣
↪variable
```

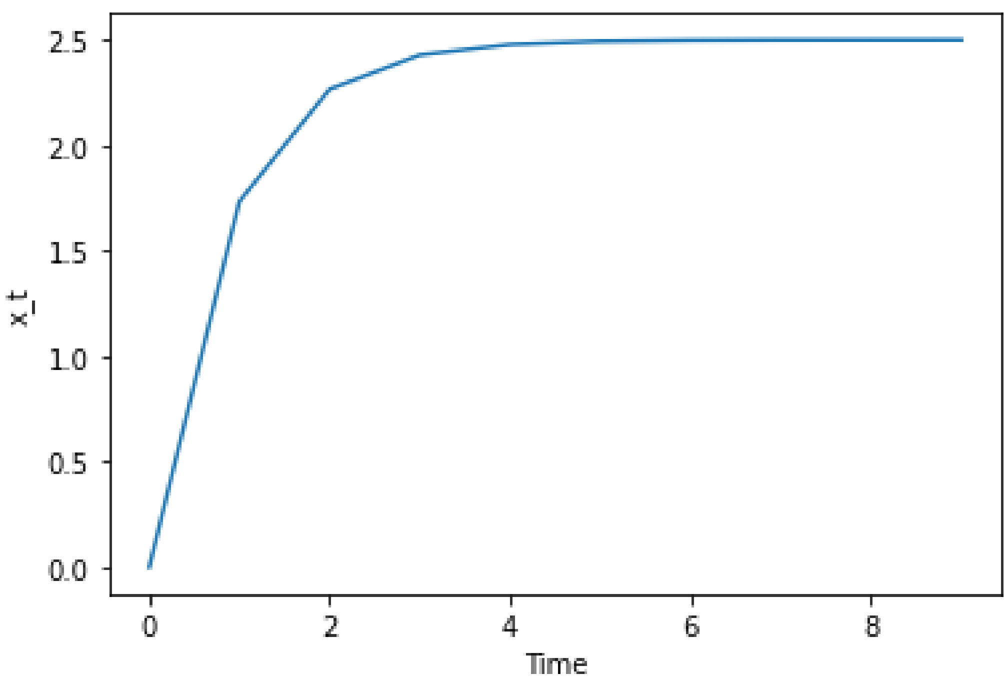
```
[0, 1.7344, 2.2656, 2.4282, 2.478, 2.4933, 2.4979, 2.4994, 2.4998, 2.4999, 2.5,
2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5,
2.5, 2.5, 2.5, 2.5, 2.5]
```

Note that 2.5 = (10 + -5) / 2.

[23]:
```
plt.plot(range(len(xs[0])), [a.item() + 10 for a in xs[0]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```

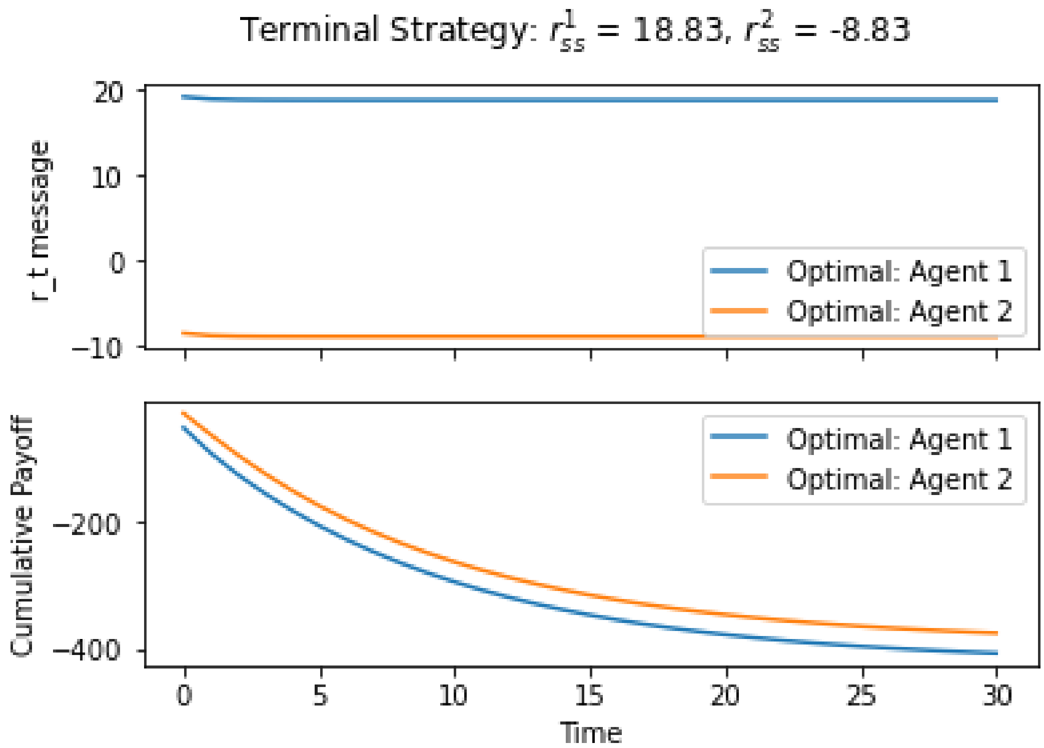


[24]:
```
plt.plot(range(10), [a.item() + 10 for a in xs[0][:10]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



3.4 3. Let $x_1 = 10, x_2 = 0, x_0 = 4$. What happens now?

[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 # 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]
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, num_agents = 2, set_cap = 1000)
```



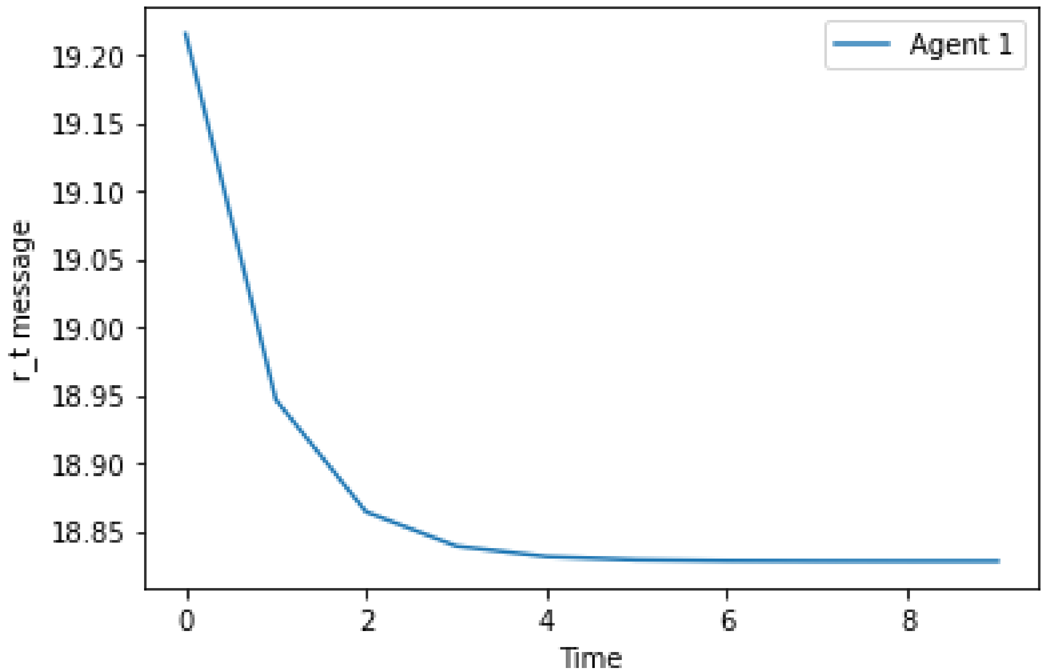
[26]:
```
print([round(a.item() + 10, 4) for a in rs[0]]) # r1
```

```
[19.2152, 18.9464, 18.864, 18.8388, 18.8311, 18.8287, 18.828, 18.8278, 18.8277,
18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277,
18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277, 18.8277,
18.8277, 18.8277, 18.8277, 18.8277]
```

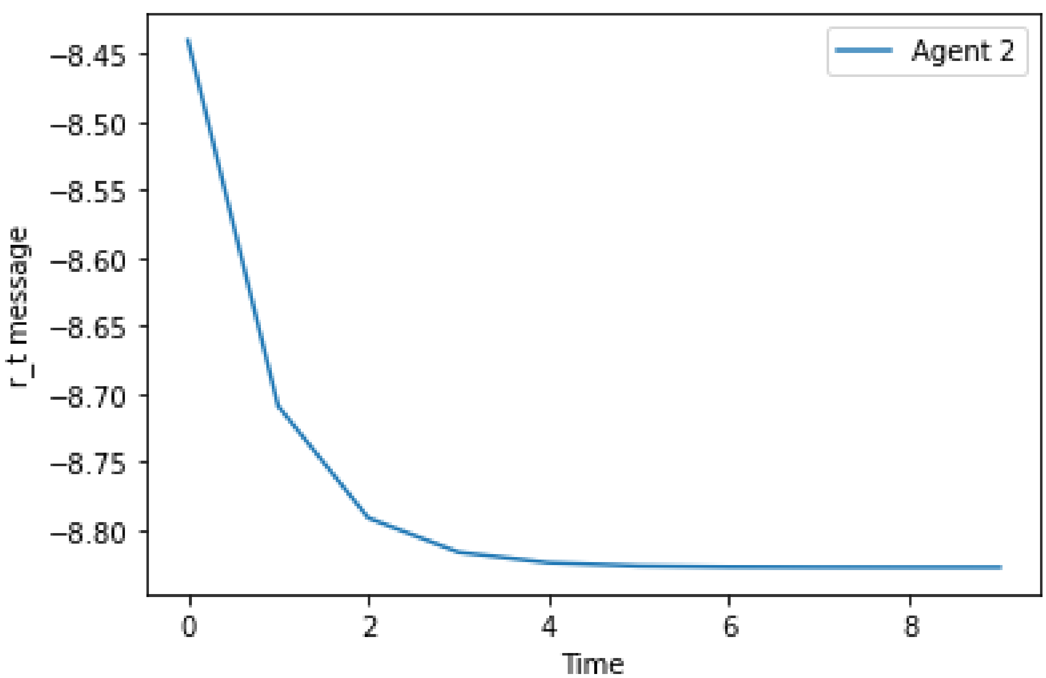
[27]:
```
print([round(a.item(), 4) for a in rs[1]]) # r2
```

```
[-8.4401, -8.709, -8.7913, -8.8166, -8.8243, -8.8266, -8.8274, -8.8276, -8.8277,
-8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277,
-8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277, -8.8277,
-8.8277, -8.8277, -8.8277, -8.8277]
```

[28]:
```
num = 10
plt.plot(range(num), [a.item() + 10 for a in rs[0][:num]], label = "Agent 1")
plt.xlabel("Time")
plt.ylabel("r_t message")
plt.legend()
plt.show()
```



[29]:
```
plt.plot(range(num), [a.item() for a in rs[1][:num]], label = "Agent 2")
plt.xlabel("Time")
plt.ylabel("r_t message")
plt.legend()
plt.show()
```

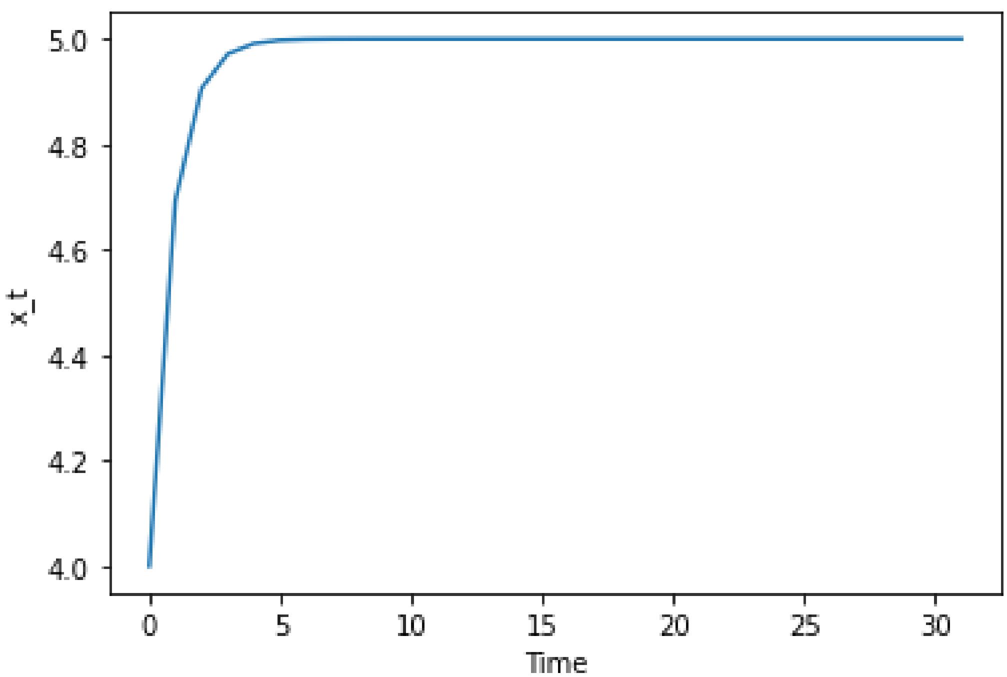


[30]:
```
print([round(a.item() + 10, 4) for a in xs[0]]) # x_t
```

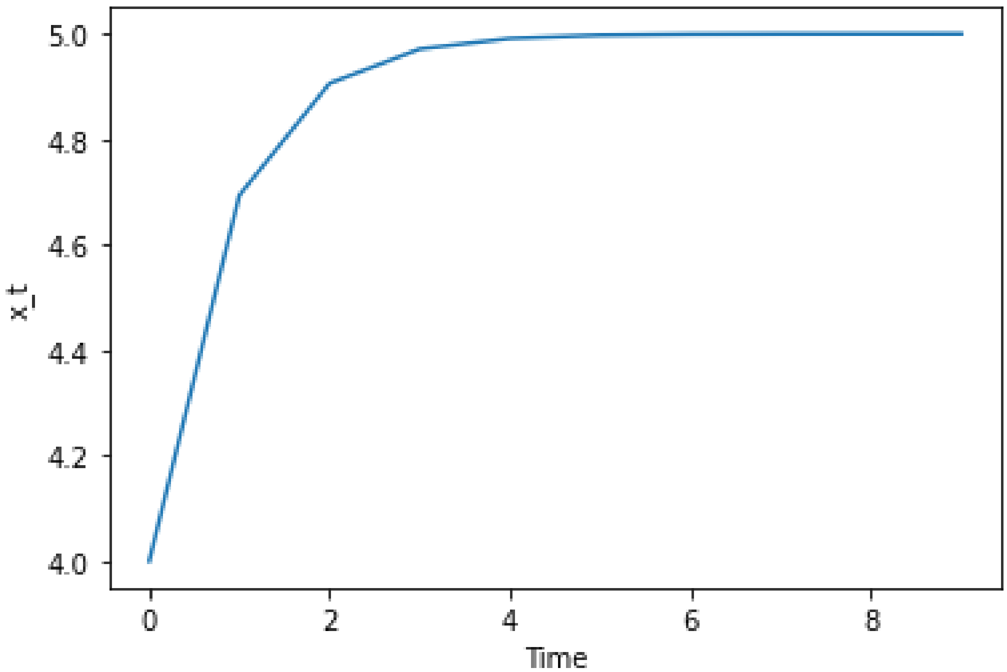
```
[4, 4.6938, 4.9062, 4.9713, 4.9912, 4.9973, 4.9992, 4.9997, 4.9999, 5.0, 5.0,
5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0,
5.0, 5.0, 5.0, 5.0, 5.0]
```

It appears to converge to exactly in between the two agendas.

[31]:
```
plt.plot(range(len(xs[0])), [a.item() + 10 for a in xs[0]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



[32]:
```
plt.plot(range(10), [a.item() + 10 for a in xs[0][:10]])
plt.xlabel("Time")
plt.ylabel("x_t")
plt.show()
```



5.0
4.8
4.6
4.4
4.2
4.0
x_t
0
2
4
6
8
Time