FurtherQuestions

October 18, 2021

0.1 Further questions

James Yu, 17 October 2021

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

[2]:
```
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()
```

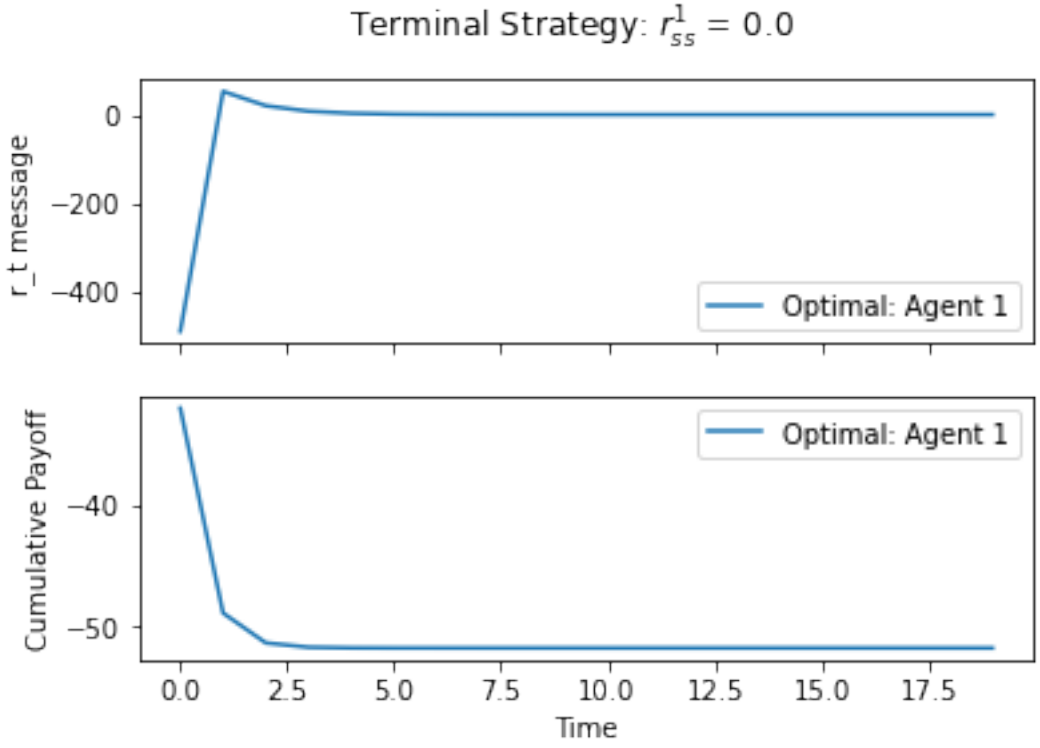
0.2 Part 5: in the short-term, which agent is better to target?

This is the stubborn agent targeting model:

[3]:
```
A = np.array([
  [0.99 * 0.99, 0.01 * 0.99],
```

```
  [0.5, 0.5],
], ndmin = 2)

B = np.array([
  [0.01],
  [0]
], ndmin = 2)

delta = 1
Q = 1 * np.identity(2)
x = np.array([
  [4],
  [4]
], ndmin = 2)

K = np.zeros((2, 2))
K_t = [Q]
K = Q
while True:
    K_new = delta * (A.T @ (K - (K @ B @ np.linalg.inv(B.T @ K @ B) @ B.T @ K))␣
 ↪@ A) + Q
    K_t.insert(0, K_new)
    current_difference = np.max(np.abs(K - K_new))
    K = K_new
    if current_difference < 10**(-14):
        break

def L_single(K_ent):
    return -1 * np.linalg.inv(B.T @ K_ent @ B) @ B.T @ K_ent @ A

x_t = x
r_ts = []

payoff = 0
payoffs = []
x_ts = [x]
i = 0
while True:
    r_t = L_single(K_t[0]) @ x_t
    r_ts.append(r_t)
    payoff += (-1 * delta**i * (x_t.T @ Q @ x_t)).item()
    payoffs.append(payoff)
    x_t_new = A @ x_t + B @ r_t
    x_ts.append(x_t_new)
    if np.max((x_t_new - x_t)**2) == 0:
        break
    x_t = x_t_new
```

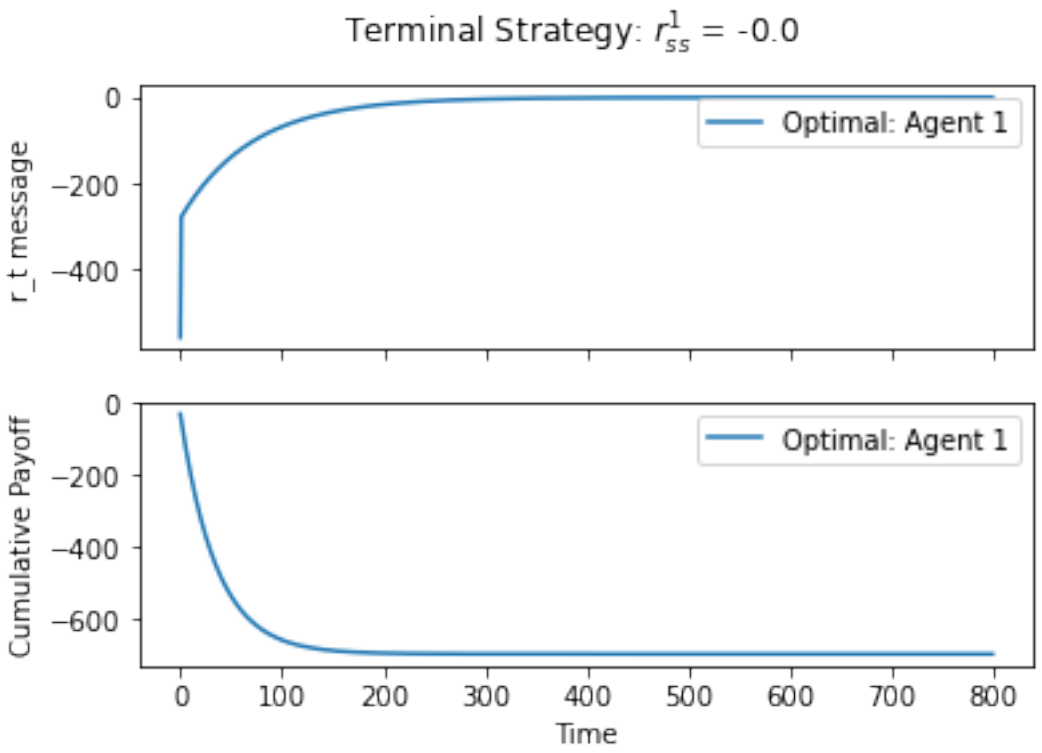
```
    i += 1

save_r_ts = r_ts
save_payoffs = payoffs
save_k = K_t
save_x = x_ts
do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 20)
```



and the easily-influenced agent targeting model:

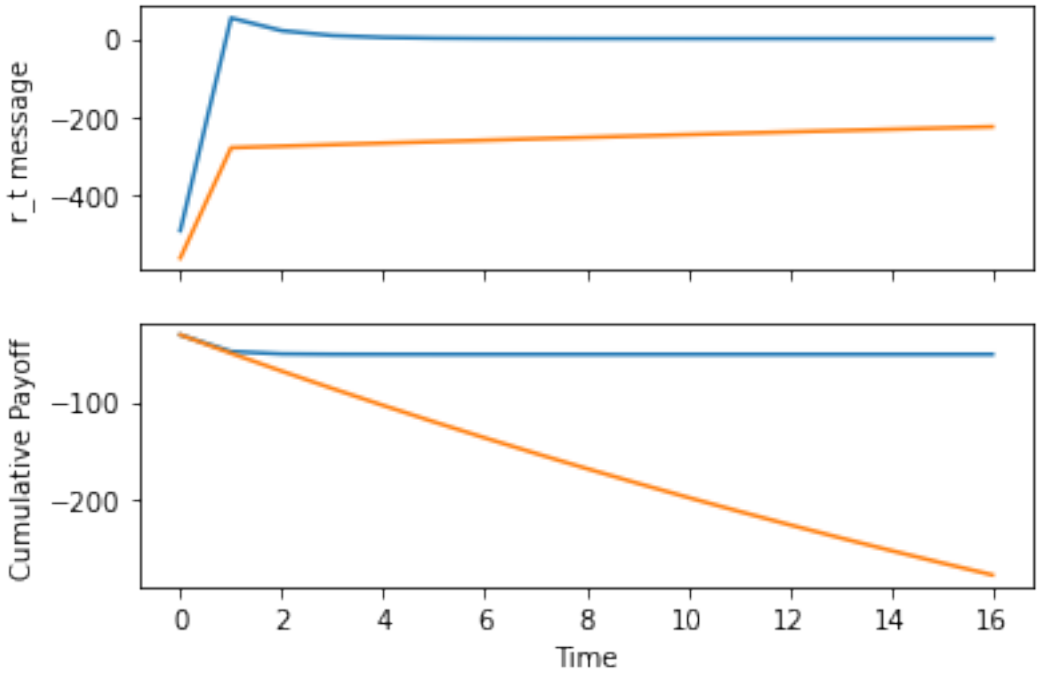
[4]:
```
A = np.array([
  [0.99, 0.01],
  [0.5 * 0.99, 0.5 * 0.99],
], ndmin = 2)

B = np.array([
  [0],
  [0.01]
], ndmin = 2)

delta = 1
Q = 1 * np.identity(2)
x = np.array([
```

```
  [4],
  [4]
], ndmin = 2)

K = np.zeros((2, 2))
K_t = [Q]
K = Q
while True:
    K_new = delta * (A.T @ (K - (K @ B @ np.linalg.inv(B.T @ K @ B) @ B.T @ K))␣
 ↪@ A) + Q
    K_t.insert(0, K_new)
    current_difference = np.max(np.abs(K - K_new))
    K = K_new
    if current_difference < 10**(-14):
        break

def L_single(K_ent):
    return -1 * np.linalg.inv(B.T @ K_ent @ B) @ B.T @ K_ent @ A

x_t = x
r_ts = []

payoff = 0
payoffs = []
x_ts = [x]
i = 0
while True:
    r_t = L_single(K_t[0]) @ x_t
    r_ts.append(r_t)
    payoff += (-1 * delta**i * (x_t.T @ Q @ x_t)).item()
    payoffs.append(payoff)
    x_t_new = A @ x_t + B @ r_t
    x_ts.append(x_t_new)
    if np.max((x_t_new - x_t)**2) == 0:
        break
    x_t = x_t_new
    i += 1


s1 = save_r_ts[0]
s2 = r_ts[0]
do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 800)
```

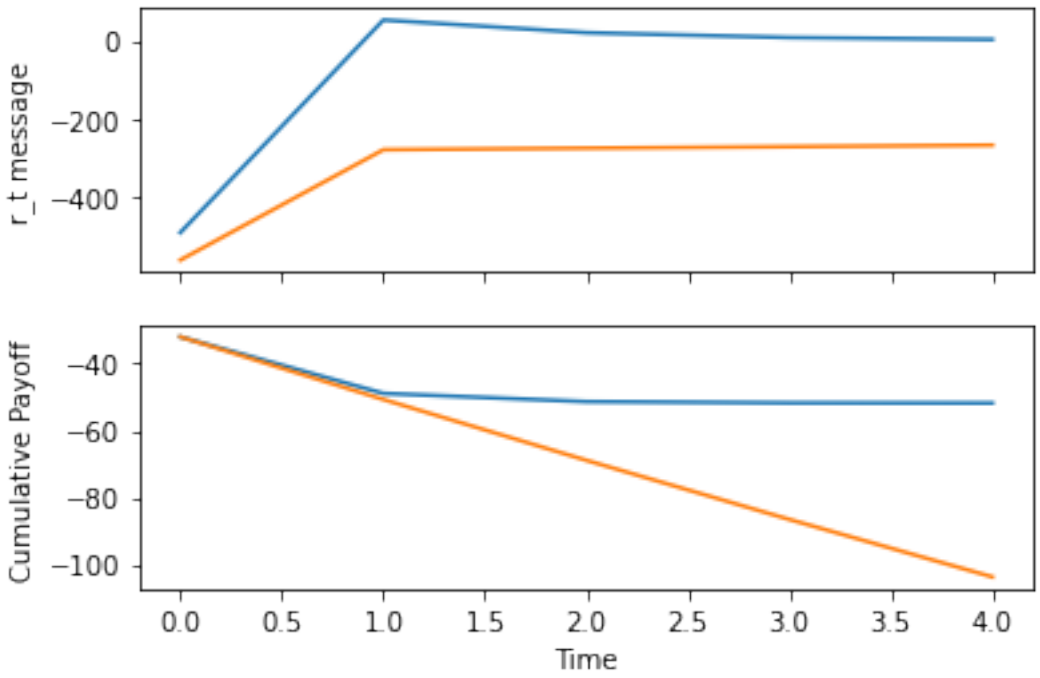


Now, if we overlay the two models, we see the following in the first 17 periods:

[5]:
```
do_plot({0:save_r_ts, 1:r_ts}, [0, 0], {0:save_payoffs, 1: payoffs}, num_agents␣
↪= 2, set_cap = 17, legend = False)
```



[6]:
```
do_plot({0:save_r_ts, 1:r_ts}, [0, 0], {0:save_payoffs, 1: payoffs}, num_agents
↪= 2, set_cap = 5, legend = False)
```



The blue line is the stubborn model and the orange line is the easily-influence model. The stubborn model upper-bounds the other model in both payoff and message.

Lets look more carefully at the short-term behaviour of the network. The opinions, for example, shift like this in the first model:

[7]:
```
save_x[:10]
```

[7]:
```
[array([[4],
        [4]]),
 array([[-0.94427191],
        [ 4.        ]]),
 array([[-0.36067977],
        [ 1.52786405]]),
 array([[-0.13776741],
        [ 0.58359214]]),
 array([[-0.05262247],
        [ 0.22291236]]),
 array([[-0.02009999],
        [ 0.08514495]]),
 array([[-0.00767751],
        [ 0.03252248]]),
 array([[-0.00293255],
        [ 0.01242248]]),
 array([[-0.00112013],
        [ 0.00474497]]),
 array([[-0.00042785],
        [ 0.00181242]])]
```

and like this in the second model:

[8]:
```
x_ts[:10]
```

[8]:
```
[array([[4],
        [4]]),
 array([[ 4.        ],
        [-1.64509196]]),
 array([[ 3.94354908],
        [-1.62187522]]),
 array([[ 3.88789484],
        [-1.59898614]]),
 array([[ 3.83302603],
        [-1.57642008]]),
 array([[ 3.77893157],
        [-1.55417249]]),
 array([[ 3.72560053],
        [-1.53223887]]),
 array([[ 3.67302213],
        [-1.5106148 ]]),
```

```
 array([[ 3.62118576],
        [-1.4892959 ]]),
 array([[ 3.57008095],
        [-1.46827787]])]
```

As expected, in the first period, the only opinion that changes is the one being targeted. In the second model, the "stubbornness" of the first naive agent is apparent, showing why the convergence time is much worse.

[9]:
```
save_payoffs[:10]
```

[9]:
```
[-32.0,
 -48.89164944001345,
 -51.35610788011103,
 -51.71566752078064,
 -51.76812656537031,
 -51.77578023682836,
 -51.77689689244506,
 -51.77705981030391,
 -51.77708357969918,
 -51.77708704760721]
```

[10]:
```
payoffs[:10]
```

[10]:
```
[-32.0,
 -50.706327568171,
 -68.88838616251601,
 -86.56086909768564,
 -103.73805788922317,
 -120.43383379476604,
 -136.66168903179027,
 -152.43473768096302,
 -167.76572628391486,
 -182.66704414399553]
```

The initial-period payoff is the same, because the opinions start out the same in both periods. But then the second model does much worse because of the extremity of the stubborn agent that isn't present in the first model.

Thus, all the properties are focused on the performance of the stubborn agent.

[11]:
```
save_r_ts[:10]
```

[11]:
```
[array([[-490.427191]]),
 array([[52.5201124]]),
 array([[20.06089784]]),
 array([[7.66258113]]),
 array([[2.92684555]]),
```

```
 array([[1.11795552]]),
 array([[0.42702101]]),
 array([[0.16310751]]),
 array([[0.06230153]]),
 array([[0.02379707]])]
```

[12]:
```
r_ts[:10]
```

[12]:
```
[array([[-560.50919634]]),
 array([[-278.7554703]]),
 array([[-274.82146964]]),
 array([[-270.94298846]]),
 array([[-267.11924324]]),
 array([[-263.34946151]]),
 array([[-259.63288169]]),
 array([[-255.96875295]]),
 array([[-252.35633508]]),
 array([[-248.79489828]])]
```

The message intensity is sustained in the second model, again because of the stubborn agent being persistent for a longer amount of time.

[13]:
```
save_k
```

[13]:
```
[array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901699, 0.30901699],
        [0.30901699, 1.30901699]]),
 array([[1.30901698, 0.30901698],
        [0.30901698, 1.30901698]]),
 array([[1.30901691, 0.30901691],
        [0.30901691, 1.30901691]]),
 array([[1.30901639, 0.30901639],
```

```
       [0.30901639, 1.30901639]]),
 array([[1.30901288, 0.30901288],
       [0.30901288, 1.30901288]]),
 array([[1.30898876, 0.30898876],
       [0.30898876, 1.30898876]]),
 array([[1.30882353, 0.30882353],
       [0.30882353, 1.30882353]]),
 array([[1.30769231, 0.30769231],
       [0.30769231, 1.30769231]]),
 array([[1.3, 0.3],
       [0.3, 1.3]]),
 array([[1.25, 0.25],
       [0.25, 1.25]]),
 array([[1., 0.],
       [0., 1.]])]
```

The above is the K sequence for the first model. For the second model, it ends with:

[14]:
```
K_t[-20:]
```

[14]:
```
[array([[16.49134411,  0.15647822],
       [ 0.15647822,  1.00158059]]),
 array([[15.82828149,  0.14978062],
       [ 0.14978062,  1.00151294]]),
 array([[15.14975429,  0.14292681],
       [ 0.14292681,  1.00144371]]),
 array([[14.45549894,  0.13591413],
       [ 0.13591413,  1.00137287]]),
 array([[13.74525251,  0.12873992],
       [ 0.12873992,  1.0013004 ]]),
 array([[13.01875305,  0.12140155],
       [ 0.12140155,  1.00122628]]),
 array([[12.2757399 ,  0.11389636],
       [ 0.11389636,  1.00115047]]),
 array([[11.51595406,  0.10622176],
       [ 0.10622176,  1.00107295]]),
 array([[10.73913858,  0.09837514],
       [ 0.09837514,  1.00099369]]),
 array([[9.94503893, 0.09035393],
       [0.09035393, 1.00091267]]),
 array([[9.1334034 , 0.08215559],
       [0.08215559, 1.00082985]]),
 array([[8.30398352, 0.07377761],
       [0.07377761, 1.00074523]]),
 array([[7.45653449, 0.06521752],
       [0.06521752, 1.00065876]]),
 array([[6.59081566, 0.05647289],
```

```
        [0.05647289, 1.00057043]]),
 array([[5.70659096, 0.04754132],
        [0.04754132, 1.00048022]]),
 array([[4.80362938, 0.0384205 ],
        [0.0384205 , 1.00038809]]),
 array([[3.8817055 , 0.02910814],
        [0.02910814, 1.00029402]]),
 array([[2.94059996, 0.01960202],
        [0.01960202, 1.000198  ]]),
 array([[1.9801, 0.0099],
        [0.0099, 1.0001]]),
 array([[1., 0.],
        [0., 1.]])]
```

and starts with:

[15]:
```
K_t[:20]
```

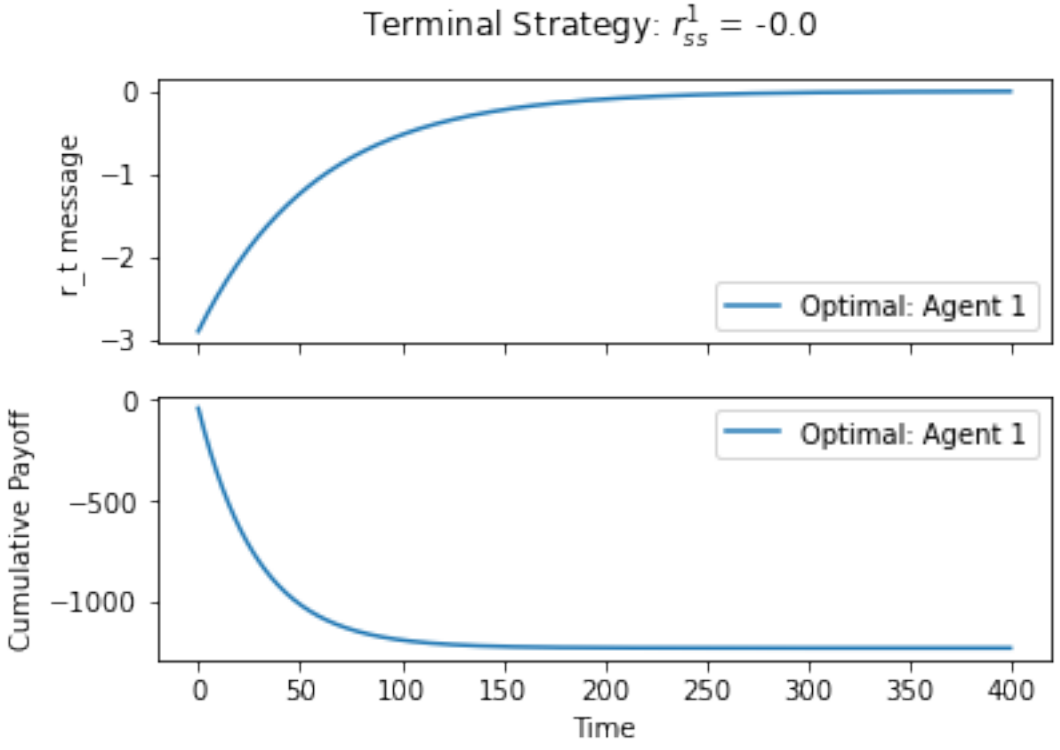
[15]:
```
[array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
        [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
```

```
       [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
       [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
       [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
       [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
       [ 0.41298866,  1.0041716 ]]),
 array([[41.88587718,  0.41298866],
       [ 0.41298866,  1.0041716 ]])]
```

Is there a difference if we impose cost? We do the following:

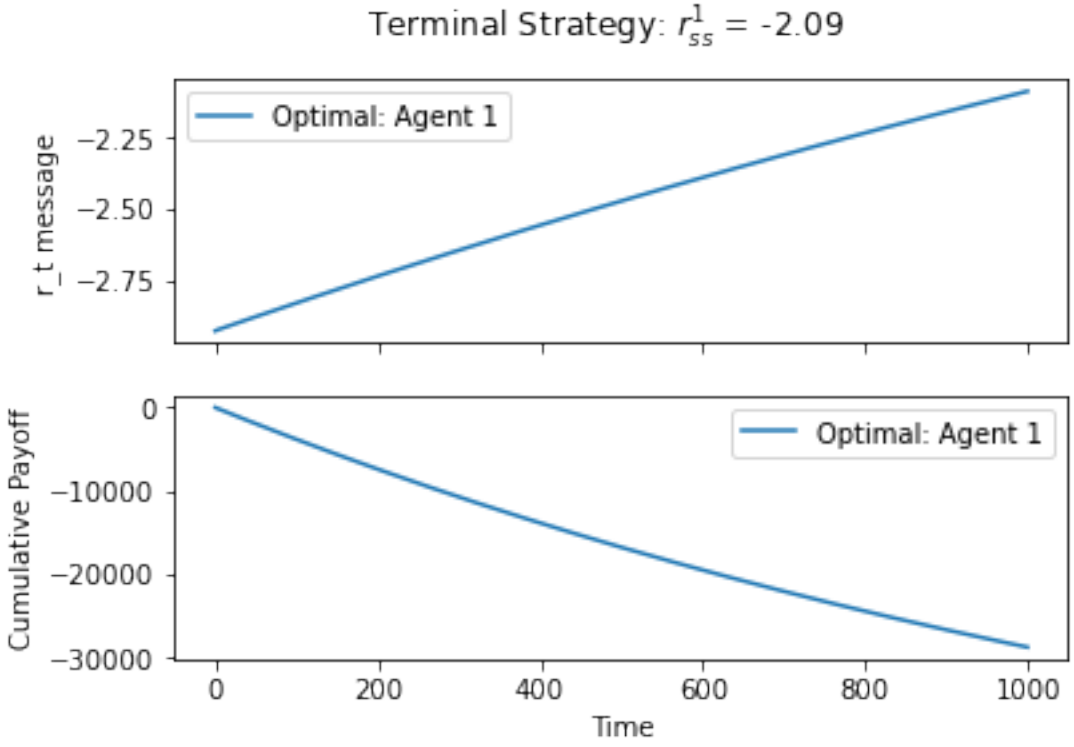
[16]:
```
A = np.array([
  [0.99 * 0.99, 0.01 * 0.99],
  [0.5, 0.5],
], ndmin = 2)

B = np.array([
  [0.01],
  [0]
], ndmin = 2)

delta = 1
Q = 1 * np.identity(2)
x = np.array([
  [4],
  [4]
], ndmin = 2)

K = np.zeros((2, 2))
K_t = [Q]
K = Q
R = 1
while True:
    K_new = delta * (A.T @ (K - (K @ B @ np.linalg.inv(B.T @ K @ B + R) @ B.T @␣
 ↪K)) @ A) + Q
    K_t.insert(0, K_new)
    current_difference = np.max(np.abs(K - K_new))
    K = K_new
    if current_difference < 10**(-14):
        break

def L_single(K_ent):
    return -1 * np.linalg.inv(B.T @ K_ent @ B + R) @ B.T @ K_ent @ A
```

```
x_t = x
r_ts = []

payoff = 0
payoffs = []
x_ts = [x]
i = 0
while True:
    r_t = L_single(K_t[0]) @ x_t
    r_ts.append(r_t)
    payoff += (-1 * delta**i * (x_t.T @ Q @ x_t)).item() + (-1 * delta**i * (R␣
 ↪* r_t**2)).item()
    payoffs.append(payoff)
    x_t_new = A @ x_t + B @ r_t
    x_ts.append(x_t_new)
    if np.max((x_t_new - x_t)**2) == 0:
        break
    x_t = x_t_new
    i += 1

save_r_ts = r_ts
save_payoffs = payoffs
save_k = K_t
save_x = x_ts
do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 400)
```



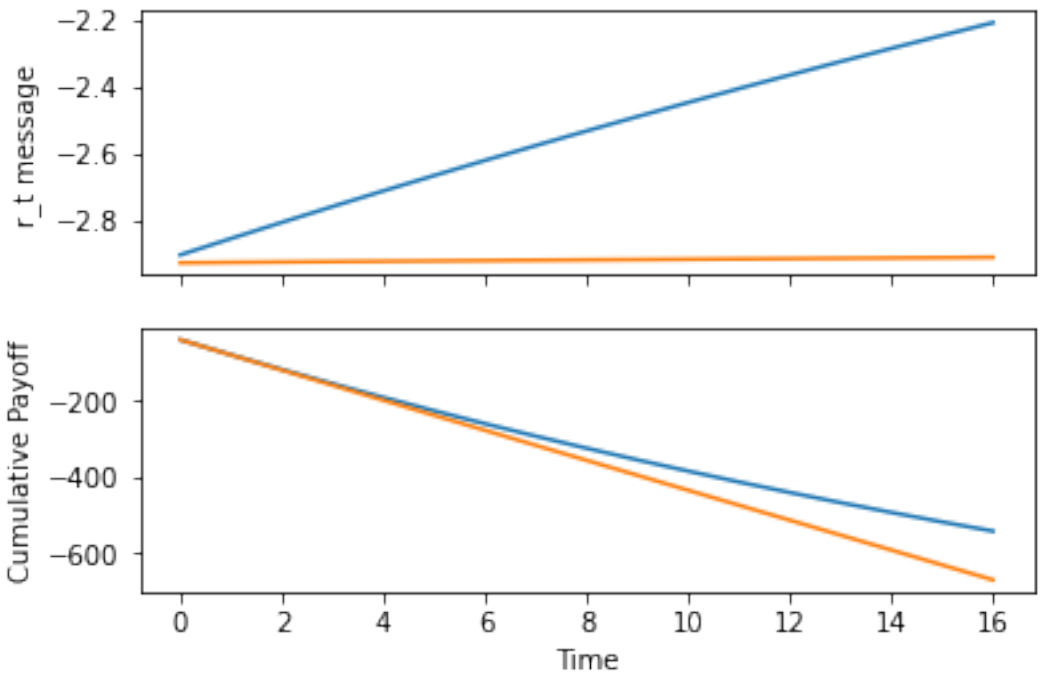
[17]:
```
A = np.array([
  [0.99, 0.01],
  [0.5 * 0.99, 0.5 * 0.99],
], ndmin = 2)

B = np.array([
  [0],
  [0.01]
], ndmin = 2)

delta = 1
Q = 1 * np.identity(2)
x = np.array([
  [4],
  [4]
], ndmin = 2)

K = np.zeros((2, 2))
K_t = [Q]
K = Q
R = 1
while True:
    K_new = delta * (A.T @ (K - (K @ B @ np.linalg.inv(B.T @ K @ B + R) @ B.T @ K)) @ A) + Q
    K_t.insert(0, K_new)
    current_difference = np.max(np.abs(K - K_new))
    K = K_new
    if current_difference < 10**(-14):
        break

def L_single(K_ent):
    return -1 * np.linalg.inv(B.T @ K_ent @ B + R) @ B.T @ K_ent @ A

x_t = x
r_ts = []

payoff = 0
payoffs = []
x_ts = [x]
i = 0
while True:
    r_t = L_single(K_t[0]) @ x_t
    r_ts.append(r_t)
    payoff += (-1 * delta**i * (x_t.T @ Q @ x_t)).item() + (-1 * delta**i * (R * r_t**2)).item()
```

```
    payoffs.append(payoff)
    x_t_new = A @ x_t + B @ r_t
    x_ts.append(x_t_new)
    if abs(np.max((x_t_new - x_t)**2)) < 10**(-14):
        break
    x_t = x_t_new
    i += 1

do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 1000)
```



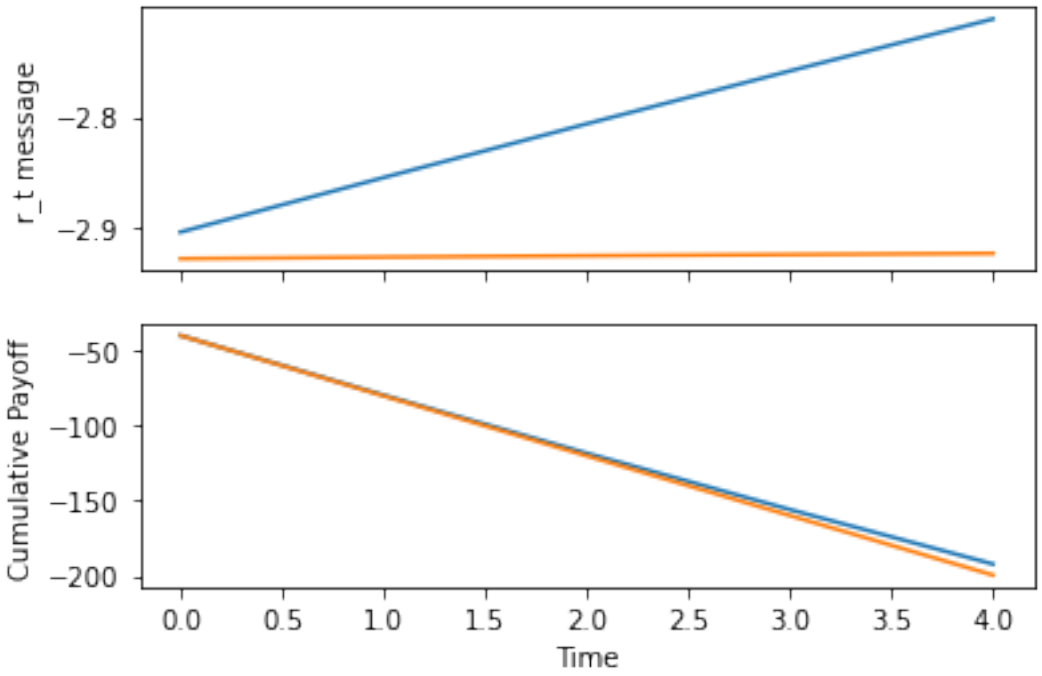
Note that is not the convergence point, the amount of periods required to converge is just very very high. If we compare the short-term:

[18]:
```
do_plot({0:save_r_ts, 1:r_ts}, [0, 0], {0:save_payoffs, 1: payoffs}, num_agents␣
↪= 2, set_cap = 17, legend = False)
```



Once again, blue is the stubborn model and orange is the other model. This seems to exhibit similar behaviour.

[19]:
```
do_plot({0:save_r_ts, 1:r_ts}, [0, 0], {0:save_payoffs, 1: payoffs}, num_agents␣
↪= 2, set_cap = 5, legend = False)
```



[20]:
```
save_payoffs[:5]
```

[20]:
```
[-40.43011835587059,
 -80.03100850447103,
 -118.56052365544508,
 -155.92206791518328,
 -192.09038629829087]
```

[21]:
```
payoffs[:5]
```

[21]:
```
[-40.568865499903104,
 -80.58004022277242,
 -120.31039914893665,
 -159.89168231226228,
 -199.38718973192456]
```

Payoff in the second model is universally worse.

0.2.1 Part 6: behaviour of K_1, K_2, K_3 in the stubborn case

Intuition tells me all three should be decreasing with time because $K_T = c * I_n$, meaning $K_{2T} = 0$ and we know K_{20} is not zero so it must be decreasing at least at some point.

[22]:
```
from sympy import *

K1, K2, K3, a11, a12 = symbols("K1 K2 K3 a11 a12")

K = Matrix(([[K1, K2], [K2, K3]]))
Q = 0.5 * eye(2) # in the email you mentioned 0.5 I_n
A = Matrix(([[a11, a12], [0.5, 0.5]]))
B = Matrix(([[1 - a11 - a12], [0]]))
# note there is no cost here since K_1 \neq K_3 when R > 0
K
```

[22]:
$$\begin{bmatrix} K_1 & K_2 \\ K_2 & K_3 \end{bmatrix}$$

[23]:
```
def solveK(K_choice):
    return simplify(A.T *(K_choice - (K_choice*B*(B.T * K_choice * B).inv() * B.
 ↪T * K_choice)) * A + Q)
K_sol = solveK(Q)
K_sol
```

[23]:
$$\begin{bmatrix} 0.625 & 0.125 \\ 0.125 & 0.625 \end{bmatrix}$$

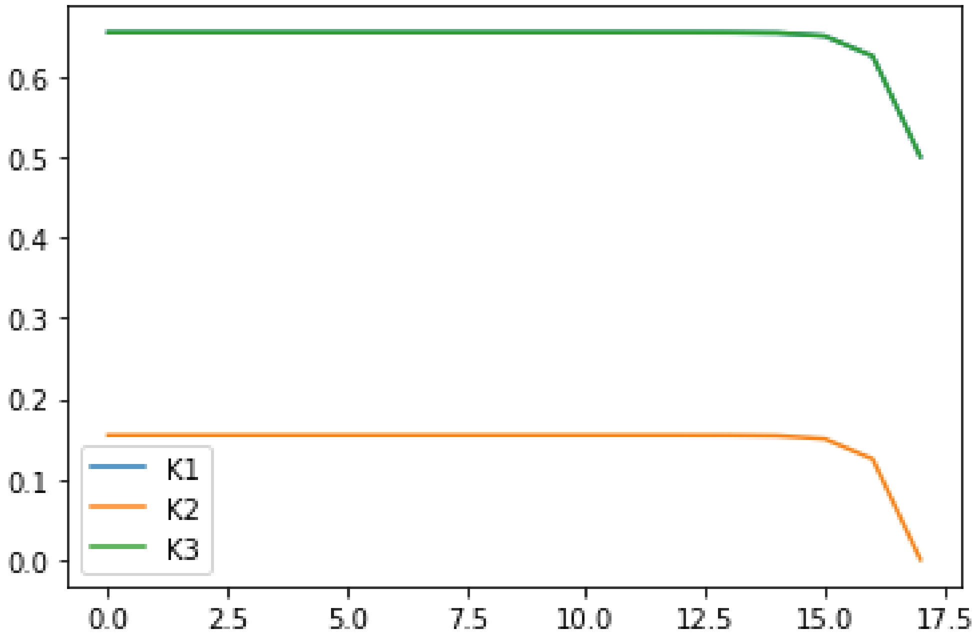
[24]:
```
solveK(K_sol)
```

[24]:
$$\begin{bmatrix} 0.65 & 0.15 \\ 0.15 & 0.65 \end{bmatrix}$$

Note we proved that the optimal solution is invariant of the A matrix due to the nature of the problem, so in this specific case, we can just plot the sequences from the numerical version.

[25]:
```
A = np.array([
  [0.99 * 0.99, 0.01 * 0.99],
  [0.5, 0.5],
], ndmin = 2)

B = np.array([
  [0.01],
  [0]
], ndmin = 2)

delta = 1
Q = 0.5 * np.identity(2)
x = np.array([
  [4],
  [4]
], ndmin = 2)

K = np.zeros((2, 2))
K_t = [Q]
K = Q
while True:
    K_new = delta * (A.T @ (K - (K @ B @ np.linalg.inv(B.T @ K @ B) @ B.T @ K))␣
 ↪@ A) + Q
    K_t.insert(0, K_new)
    current_difference = np.max(np.abs(K - K_new))
    K = K_new
    if current_difference < 10**(-14):
        break

K1s = []
K2s = []
K3s = []
for Kt in K_t:
    K1s.append(Kt[0,0])
    K2s.append(Kt[0,1])
    K3s.append(Kt[1,1])

plt.plot(range(len(K_t)), K1s, label = "K1")
plt.plot(range(len(K_t)), K2s, label = "K2")
plt.plot(range(len(K_t)), K3s, label = "K3")
```

```
plt.legend()
plt.show()
```



They are in fact decreasing with time, which can be seen in the following sequence:

[26]:
```
K_t
```

[26]:
```
[array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.6545085, 0.1545085],
        [0.1545085, 0.6545085]]),
 array([[0.65450849, 0.15450849],
        [0.15450849, 0.65450849]]),
 array([[0.65450845, 0.15450845],
```

```
       [0.15450845, 0.65450845]]),
 array([[0.6545082, 0.1545082],
       [0.1545082, 0.6545082]]),
 array([[0.65450644, 0.15450644],
       [0.15450644, 0.65450644]]),
 array([[0.65449438, 0.15449438],
       [0.15449438, 0.65449438]]),
 array([[0.65441176, 0.15441176],
       [0.15441176, 0.65441176]]),
 array([[0.65384615, 0.15384615],
       [0.15384615, 0.65384615]]),
 array([[0.65, 0.15],
       [0.15, 0.65]]),
 array([[0.625, 0.125],
       [0.125, 0.625]]),
 array([[0.5, 0. ],
       [0. , 0.5]])]
```

Again, we are able to use the numerical result because this is a special case where A does not factor because we pre-specified enough of the structure.