Solve_Improved

October 17, 2021

0.1 Symbolic solutions and some numerical checking

James Yu, 17 October 2021

[1]:
```
from sympy import *
```

0.2 Part 1: a symbolic proof of the simple targeting second agent result with better equations

[2]:
```
K1, K2, K3, a1, a2 = symbols("K1 K2 K3 a1 a2")

K = Matrix(([[K1, K2], [K2, K3]]))
Q = eye(2)
A = Matrix(([[a1, 1 - a1], [a2, a2]]))
B = Matrix(([[0], [1 - 2*a2]]))
K
```

[2]:

$$\begin{bmatrix} K_1 & K_2 \\ K_2 & K_3 \end{bmatrix}$$

[3]:
```
A
```

[3]:

$$\begin{bmatrix} a_1 & 1 - a_1 \\ a_2 & a_2 \end{bmatrix}$$

[4]:
```
B
```

[4]:

$$\begin{bmatrix} 0 \\ 1 - 2a_2 \end{bmatrix}$$

If we plug these into the equation directly, we obtain:

[5]:
```
K_sol = simplify(A.T *(K - K*B*(B.T * K * B).inv() * B.T * K) * A + Q)
K_sol
```

[5]:

$$\begin{bmatrix} K_1 a_1^2 - \frac{K_2^2 a_1^2}{K_3} + 1 & \frac{a_1\left(-K_1 K_3 a_1 + K_1 K_3 + K_2^2 a_1 - K_2^2\right)}{K_3} \\ \frac{a_1\left(-K_1 K_3 a_1 + K_1 K_3 + K_2^2 a_1 - K_2^2\right)}{K_3} & K_1 a_1^2 - 2K_1 a_1 + K_1 - \frac{K_2^2 a_1^2}{K_3} + \frac{2K_2^2 a_1}{K_3} - \frac{K_2^2}{K_3} + 1 \end{bmatrix}$$

This leads to a system of three equations in three unknowns because of the symmetry, so we need to solve these all at the same time.

$$K_1 = (a_1 - 0)^2(K_1 - \frac{K_2^2}{K_3}) + 1$$

$$K_3 = (a_1 - 1)^2(K_1 - \frac{K_2^2}{K_3}) + 1$$

$$K_2K_3 = (K_1K_3 - K_2^2)a_1(1 - a_1)$$

Notice that $K_1K_3 - K_2^2$ is the determinant of K.

From here, we can use the common terms in the first two expressions to say:

$$\frac{K_1 - 1}{(a_1 - 0)^2} = \frac{K_3 - 1}{(a_1 - 1)^2}$$

[6]:
```
LHS = (K1 - 1)/(a1**2)
RHS = (K3 - 1)/((a1 - 1)**2)
K1_of_K3 = simplify(expand(solve(LHS - RHS, K1)[0]))
K1_of_K3
```

[6]: $$\frac{K_3a_1^2 - 2a_1 + 1}{a_1^2 - 2a_1 + 1}$$

We can also solve for K_1 in the first expression.

[7]:
```
K1_of_K1K2K3 = (a1**2)*(K1 - (K2**2)/K3) + 1
K1_of_K2K3 = solve(K1_of_K1K2K3 - K1, K1)[0]
K1_of_K2K3
```

[7]: $$\frac{K_2^2a_1^2 - K_3}{K_3\left(a_1^2 - 1\right)}$$

and then equate these to solve for K_2^2:

[8]:
```
K2squared_of_K3 = solve(K1_of_K3 - K1_of_K2K3, K2**2)[0]
K2squared_of_K3
```

[8]: $$\frac{K_3\left(K_3a_1 + K_3 - 2\right)}{a_1 - 1}$$

[9]:
```
K2K3_temp = (K1*K3 - K2**2)*a1*(1-a1)
K2K3 = simplify(K2K3_temp.subs(K2**2, K2squared_of_K3))
K2K3
```

[9]: $$K_3a_1\left(-K_1a_1 + K_1 + K_3a_1 + K_3 - 2\right)$$

The above can actually be further simplified to the expression:

$$K_2 = a_1(K_1(1 - a_1) + K_3(1 + a_1) - 2)$$

[10]:
```
K2_of_K3 = K2K3.subs(K1, K1_of_K3) / K3
simplify(K2_of_K3)
```

[10]:
$$\frac{a_1(1 - K_3)}{a_1 - 1}$$

This is a surprisingly nice expression. We now have expressions relating K_3 to K_2 and K_1 to K_3.

This system is:

$$K_2 = \frac{a_1(1 - K_3)}{a_1 - 1}$$

$$K_1 = \frac{K_3 a_1^2 - 2a_1 + 1}{(a_1 - 1)^2}$$

The last step is using the second of the original three expressions to solve for K_3.

[11]:
```
K3_of_K1K2K3 = ((a1 - 1)**2)*(K1 - (K2**2)/K3) + 1
K3_tosolve = simplify(K3_of_K1K2K3.subs(K1, K1_of_K3).subs(K2, K2_of_K3))
K3_tosolve
```

[11]:
$$2a_1^2 - 2a_1 + 2 - \frac{a_1^2}{K_3}$$

[12]:
```
possibly_K3 = solve(K3_tosolve - K3, K3)[0]
possibly_K3
```

[12]:
$$a_1^2 - a_1 + (1 - a_1)\sqrt{a_1^2 + 1} + 1$$

[13]:
```
possibly_not_K3 = solve(K3_tosolve - K3, K3)[1]
possibly_not_K3
```

[13]:
$$a_1^2 - a_1 + (a_1 - 1)\sqrt{a_1^2 + 1} + 1$$

K_2 if K_3 is positive:

[14]:
```
possibly_K2 = simplify(K2_of_K3.subs(K3, possibly_K3))
possibly_K2
```

[14]:
$$a_1\left(-a_1 + \sqrt{a_1^2 + 1}\right)$$

K_1 if K_3 is positive:

[15]:
```
possibly_K1 = simplify(K1_of_K3.subs(K3, possibly_K3))
possibly_K1
```

[15]:
$$\frac{a_1^3 - a_1^2\sqrt{a_1^2 + 1} + a_1 - 1}{a_1 - 1}$$

K_2 if K_3 is negative:

[16]:
```
possibly_not_K2 = simplify(K2_of_K3.subs(K3, possibly_not_K3))
possibly_not_K2
```

[16]:

$$-a_1\left(a_1+\sqrt{a_1^2+1}\right)$$

K_1 if K_3 is negative:

[17]:
```
possibly_not_K1 = simplify(K1_of_K3.subs(K3, possibly_not_K3))
possibly_not_K1
```

[17]:

$$\frac{a_1^3+a_1^2\sqrt{a_1^2+1}+a_1-1}{a_1-1}$$

The next question to answer is whether K has a positive or negative K_2. Recall that we start with $K_T = Q = cI$ meaning the diagonal is positive and the off-diagonal is zero.

What follows is a proof by induction using $K_T = Q$ as the base case:

[18]:
```
K_TM1 = simplify(A.T *(Q - Q*B*(B.T * Q * B).inv() * B.T * Q) * A + Q)
K_TM1
```

[18]:

$$\begin{bmatrix} a_1^2+1 & a_1(1-a_1) \\ a_1(1-a_1) & a_1^2-2a_1+2 \end{bmatrix}$$

K_1 must be positive and K_2 must be non-negative, and K_3 simplifies to $(a_1-1)^2+1$ which is also positive.

Also, $K_2 < K_1$ and $K_2 < K_3$ holds since we know $K_2 < 1$, $K_1 \geq 1$, $K_3 \geq 1$ all hold. So $det(K) > 0$.

Now, the induction step. Suppose K_t has a positive diagonal and non-negative off-diagonal such that $det(K_t) > 0$.

$$K_{1t-1} = (a_1-0)^2(K_{1t} - \frac{K_{2t}^2}{K_{3t}}) + 1$$

We want to determine if $K_1 > K_2^2/K_3$. Since we know $det(K_t) > 0$, it follows that $K_1K_3 > K_2^2$. Thus, since $K_3 > 0$, we know $K_1 > K_2^2/K_3$, so $K_{1t-1} > 0$.

$$K_{3t-1} = (a_1-1)^2(K_{1t} - \frac{K_{2t}^2}{K_{3t}}) + 1$$

The same conclusion results in $K_{3t-1} > 0$.

$$K_{2t-1} = \frac{(K_{1t}K_{3t} - K_{2t}^2)a_1(1-a_1)}{K_{3t}}$$

Finally, the same conclusion, combined with $K_3 > 0$, implies $K_2 \geq 0$.

Now, we wish to check if $det(K_{t-1}) > 0$. This requires checking $K_1K_3 > K_2^2$ in K_t.

[19]:
```
K1tm1 = a1**2 * (K1 - K2**2 / K3) + 1
K1tm1
```

[19]:

$$a_1^2\left(K_1 - \frac{K_2^2}{K_3}\right) + 1$$

[20]:
```
K3tm1 = (a1-1)**2 * (K1 - K2**2 / K3) + 1
K3tm1
```

[20]:

$$\left(K_1 - \frac{K_2^2}{K_3}\right)(a_1 - 1)^2 + 1$$

[21]:
```
EXPR1 = simplify(K1tm1 * K3tm1)
EXPR1
```

[21]:

$$\frac{\left(K_3 + a_1^2\left(K_1K_3 - K_2^2\right)\right)\left(K_3 + (a_1 - 1)^2\left(K_1K_3 - K_2^2\right)\right)}{K_3^2}$$

[22]:
```
K2tm1 = (K1 * K3 - K2**2)*a1*(1-a1)/K3
K2tm1
```

[22]:

$$\frac{a_1(1 - a_1)\left(K_1K_3 - K_2^2\right)}{K_3}$$

[23]:
```
EXPR2 = K2tm1 * K2tm1
EXPR2
```

[23]:

$$\frac{a_1^2(1 - a_1)^2\left(K_1K_3 - K_2^2\right)^2}{K_3^2}$$

Since $K_3 > 0$ we just need to compare the numerators.

[24]:
```
DETERMINANT_SIGNED = expand(fraction(EXPR1)[0]) - expand(fraction(EXPR2)[0])
DETERMINANT_SIGNED
```

[24]:

$$2K_1K_3^2a_1^2 - 2K_1K_3^2a_1 + K_1K_3^2 - 2K_2^2K_3a_1^2 + 2K_2^2K_3a_1 - K_2^2K_3 + K_3^2$$

The above is EXPR1, which is $K_1 * K_3$, minus EXPR2, which is K_2^2, so we have $K_1K_3 - K_2^2$. If this is positive, then the determinant is positive.

[25]:
```
simplify(DETERMINANT_SIGNED / K3)
```

[25]:

$$2K_1K_3a_1^2 - 2K_1K_3a_1 + K_1K_3 - 2K_2^2a_1^2 + 2K_2^2a_1 - K_2^2 + K_3$$

Rearranging this expression gives

$$K_1K_3a_1^2 + K_1K_3(a_1^2 - 2a_1 + 1) - (2K_2^2a_1^2 - 2K_2a_1 + K_2^2) + K_3$$

which is

$$K_1K_3a_1^2 + K_1K_3(a_1-1)^2 - K_2^2a_1^2 - K_2^2(a_1-1)^2 + K_3$$

which simplifies to

$$a_1^2(K_1K_3 - K_2^2) + (a_1-1)^2(K_1K_3 - K_2^2) + K_3$$

which is positive, since $K_1K_3 - K_2^2 > 0$ is given. Thus, $det(K_{t-1}) > 0$, and the proof is complete.

It follows that $K_1 > 0, K_2 \geq 0, K_3 > 0$ holds for all t.

Thus, we take the $K_3 > 0$ case.

Note that $K_2 = 0$ only happens when $a_1 = 1$ or $a_1 = 0$, in which case the system is not fully connected. Otherwise, $K_2 > 0$ holds by the above process.

We can also double-check that the expression is correct:

[26]:
```
K_steady = Matrix(([[possibly_K1, possibly_K2], [possibly_K2, possibly_K3]]))
K_steady
```

[26]:

$$\begin{bmatrix} \frac{a_1^3 - a_1^2\sqrt{a_1^2+1} + a_1 - 1}{a_1 - 1} & a_1\left(-a_1 + \sqrt{a_1^2+1}\right) \\ a_1\left(-a_1 + \sqrt{a_1^2+1}\right) & a_1^2 - a_1 + (1-a_1)\sqrt{a_1^2+1} + 1 \end{bmatrix}$$

[27]:
```
K_steady.subs(a1, 0.5)
```

[27]:

$$\begin{bmatrix} 1.30901699437495 & 0.309016994374947 \\ 0.309016994374947 & 1.30901699437495 \end{bmatrix}$$

[28]:
```
K_check = simplify(A.T *(K_steady - K_steady*B*(B.T * K_steady * B).inv() * B.T␣
↪* K_steady) * A + Q)
K_check
```

[28]:

$$\begin{bmatrix} \frac{2a_1^5-2a_1^4\sqrt{a_1^2+1}-2a_1^4+2a_1^3\sqrt{a_1^2+1}+3a_1^3-2a_1^2\sqrt{a_1^2+1}-3a_1^2+2a_1\sqrt{a_1^2+1}+2a_1-\sqrt{a_1^2+1}-1}{a_1^3-a_1^2\sqrt{a_1^2+1}-2a_1^2+2a_1\sqrt{a_1^2+1}+2a_1-\sqrt{a_1^2+1}-1} & \frac{a_1(-2a_1^3+2a_1^2\sqrt{a_1^2+1}+2a_1^2-2a_1\sqrt{a_1^2+1}-2a_1}{a_1^2-a_1\sqrt{a_1^2+1}-a_1+\sqrt{a_1^2+1}+1} \\ \frac{a_1\left(-2a_1^3+2a_1^2\sqrt{a_1^2+1}+2a_1^2-2a_1\sqrt{a_1^2+1}-2a_1+\sqrt{a_1^2+1}+1\right)}{a_1^2-a_1\sqrt{a_1^2+1}-a_1+\sqrt{a_1^2+1}+1} & \frac{2a_1^4-2a_1^3\sqrt{a_1^2+1}-4a_1^3+4a_1^2\sqrt{a_1^2+1}+5a_1^2-4a_1\sqrt{a_1^2+}}{a_1^2-a_1\sqrt{a_1^2+1}-a_1+\sqrt{a_1^2+1}+1} \end{bmatrix}$$

[29]:
```
K_check.subs(a1, 0.5)
```

[29]:

$$\begin{bmatrix} 1.30901699437495 & 0.309016994374947 \\ 0.309016994374947 & 1.30901699437495 \end{bmatrix}$$

[30]:
```
import numpy as np
for res in np.linspace(0, 0.99999999, 500):
    print(np.allclose(np.array(K_steady.subs(a1, res)).astype(np.float64), np.
↪array(K_check.subs(a1, res)).astype(np.float64)), end = " ")
```

```
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True True True True True True True True True True True True True
True True True True
```

This says that, for 500 evenly-spaced values of a_1 from 0 to about 1 (if its exactly 1, the system doesn't work because the stubborn agent is fixed), with the distance between values being 1/500, the solutions are always identical. This on its own is not a proof that this really is the steady-state, but it means there were no significant typos in the derivation of the steady-state matrix, so barring any other small typos, the answer should be correct.

0.2.1 Part 2: what about targeting the stubborn agent? Why is it positive?

[31]:
```
K1, K2, K3, a11, a12 = symbols("K1 K2 K3 a11 a12")

K = Matrix(([[K1, K2], [K2, K3]]))
Q = eye(2)
A = Matrix(([[a11, a12], [0.5, 0.5]]))
B = Matrix(([[1 - a11 - a12], [0]]))
K
```

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

[32]:
```
A
```

[32]: $\begin{bmatrix} a_{11} & a_{12} \\ 0.5 & 0.5 \end{bmatrix}$

[33]:
```
B
```

[33]: $\begin{bmatrix} -a_{11} - a_{12} + 1 \\ 0 \end{bmatrix}$

[34]:
```
K_sol = simplify(A.T *(K - (K*B*(B.T * K * B).inv() * B.T * K)) * A + Q)
K_sol
```

[34]: $\begin{bmatrix} 0.25K_3 + 1.0 - \frac{0.25K_2^2}{K_1} & 0.25K_3 - \frac{0.25K_2^2}{K_1} \\ 0.25K_3 - \frac{0.25K_2^2}{K_1} & 0.25K_3 + 1.0 - \frac{0.25K_2^2}{K_1} \end{bmatrix}$

So clearly here $K_1 = K_3 = K_2 + 1$.

[35]:
```
K2_of_K2 = 0.25*(K2+1) - (0.25 * K2**2)/(K2+1)
K2_of_K2
```

[35]: $$-\frac{0.25K_2^2}{K_2 + 1} + 0.25K_2 + 0.25$$

[36]:
```
solve(K2_of_K2 - K2, K2)
```

[36]:
```
[-0.809016994374947, 0.309016994374947]
```

These are the two solutions we saw in a previous notebook and now we wish to know why K_2 should be the solution on the right. Recall that $K_T = Q$.

Then:

[37]:
```
simplify(A.T *(Q - (Q*B*(B.T * Q * B).inv() * B.T * Q)) * A + Q)
```

[37]: $\begin{bmatrix} 1.25 & 0.25 \\ 0.25 & 1.25 \end{bmatrix}$

This is clearly a positive matrix with a positive determinant. If we then use a proof by induction here, assume we start with a fully positive K as above such that the determinant is positive.

[38]:
```
K_sol[1, 0]
```

[38]: $$0.25K_3 - \frac{0.25K_2^2}{K_1}$$

This is the expression for K_2 at $t-1$ as a function of K at t. This simplifies to $0.25(K_3 - K_2^2/K_1)$ and since we know $K_1 > 0$, we check the sign of $K_1K_3 - K_2^2$, which is the determinant, which is positive, so $K_2 > 0$. Since $K_1 = K_3 = K_2 + 1$, it follows that $K_1 > 0$ and $K_3 > 0$ too.

Finally, $K_1K_3 - K_2^2 = (K_2+1)^2 - K_2^2$ and since $K_2 > 1$, this is always going to be positive, so the determinant is always positive (this holds without needing to use induction). Thus, for all t, $det(K) > 0$ and all entries of K are positive. So we take the positive K_2.

0.2.2 Part 3: robustness through adding a cost term R

[39]:
```python
R = Matrix(([[symbols("R")]])) # this is a scalar written as a 1x1 matrix so␣
↪Python is happy with it
K_sol = simplify(A.T *(K - (K*B*(B.T * K * B + R).inv() * B.T * K)) * A + Q)
K_sol[0, 0]
```

[39]:
$$\frac{0.25K_1K_3a_{11}^2 + 0.5K_1K_3a_{11}a_{12} - 0.5K_1K_3a_{11} + 0.25K_1K_3a_{12}^2 - 0.5K_1K_3a_{12} + 0.25K_1K_3 + 1.0K_1Ra_{11}^2 + 1.0K_1a_1^2}{1.0K_1a_{11}^2 +}$$

The cost term increases the symbolic complexity severely.

[40]:
```python
K_sol[1, 1]
```

[40]:
$$\frac{0.25K_1K_3a_{11}^2 + 0.5K_1K_3a_{11}a_{12} - 0.5K_1K_3a_{11} + 0.25K_1K_3a_{12}^2 - 0.5K_1K_3a_{12} + 0.25K_1K_3 + 1.0K_1Ra_{12}^2 + 1.0K_1a_1^2}{1.0K_1a_{11}^2 +}$$

These (K_1 and K_3, respectively), are also no longer the same value.

[41]:
```python
K_sol[1, 0]
```

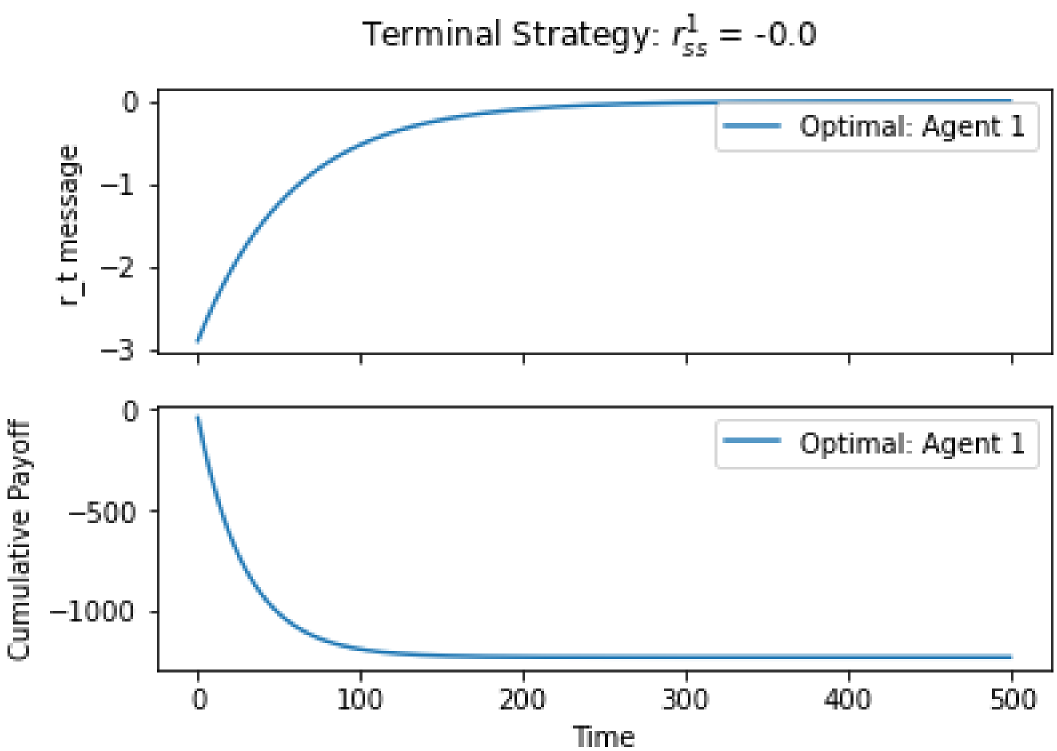
[41]:
$$\frac{0.25K_1K_3a_{11}^2 + 0.5K_1K_3a_{11}a_{12} - 0.5K_1K_3a_{11} + 0.25K_1K_3a_{12}^2 - 0.5K_1K_3a_{12} + 0.25K_1K_3 + 1.0K_1Ra_{11}a_{12} - 0.25K}{1.0K_1a_{11}^2 + 2.0K_1a_{11}a_{12} - 2.0K_1a_{11} + 1.0}$$

For now, I explore this numerically instead:

[42]:
```python
from collections import defaultdict
import matplotlib.pyplot as plt

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()

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*r_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


do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 500)
```



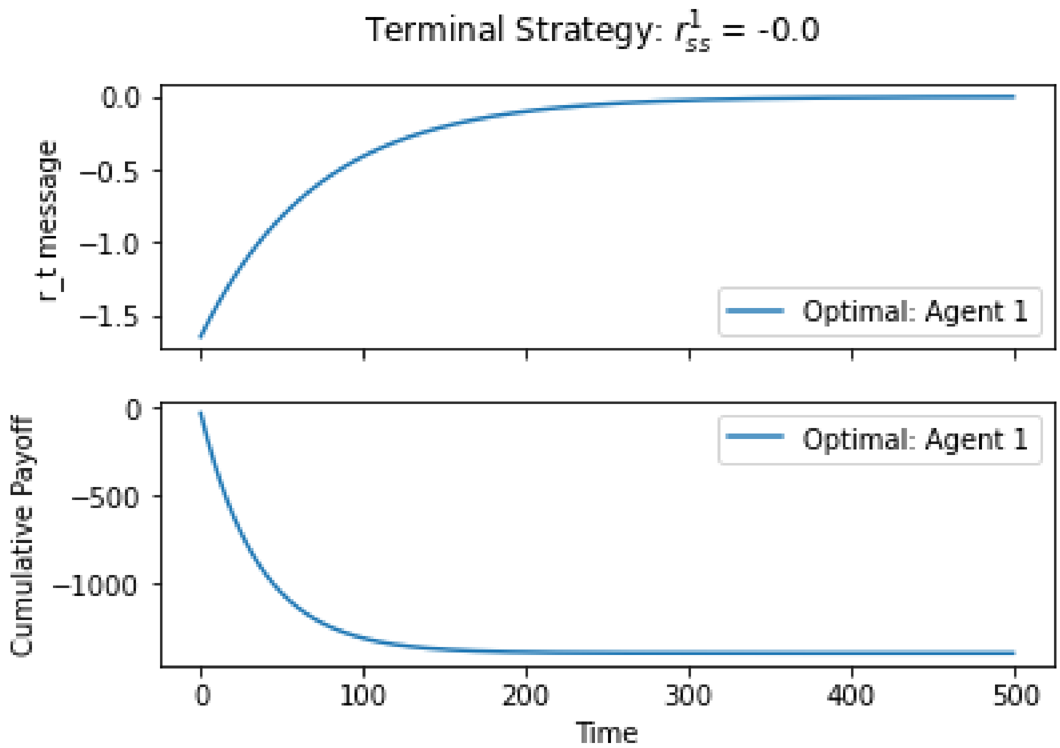
So here $R = 1$ targeting the stubborn agent. K in the steady-state is:

[43]:
```
K_t[0]
```

[43]: array([[71.79273815, 2.03297403],
       [2.03297403, 1.36915552]])

This does not have $K_1 = K_3$, but is still fully positive with positive determinant.

[44]:
```
K = np.zeros((2, 2))
K_t = [Q]
K = Q
R = 2

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

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*r_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


do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 500)
```

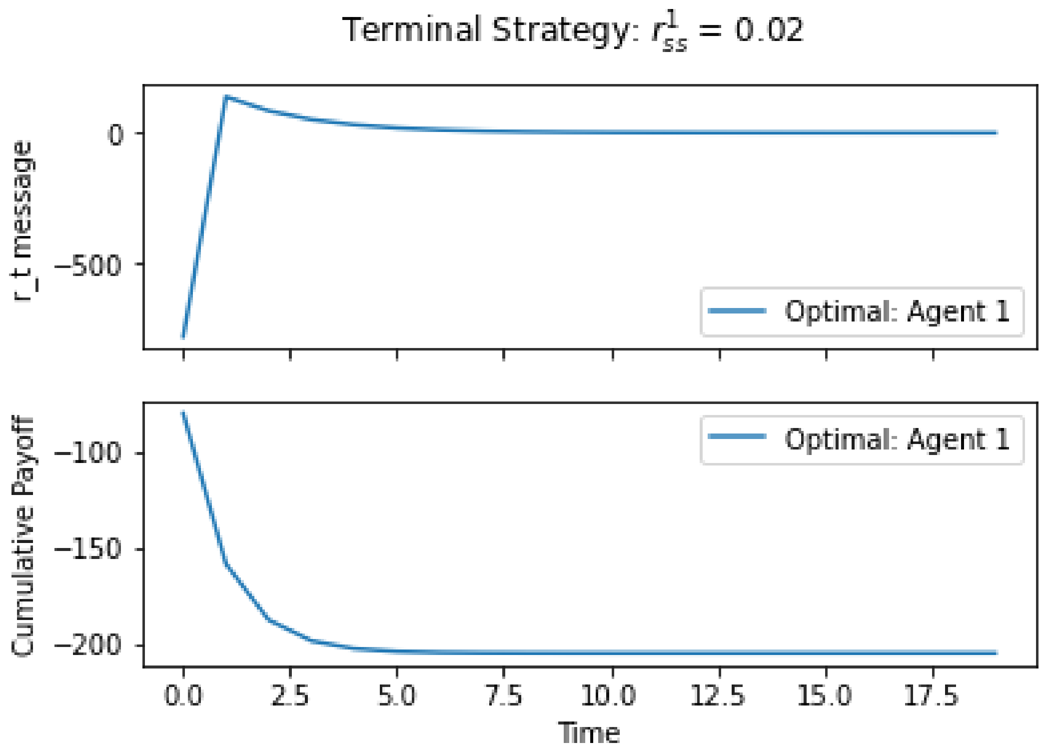


Increasing the cost brings the messages closer to zero and, compared to the $R = 0$ case from one of the previous notebooks, has longer convergence time.

0.2.3 Part 4: robustness through adding duplicate naive agents

[45]:
```
A = np.array([
  [0.99 * 0.99, 0.0025 * 0.99, 0.0025 * 0.99, 0.0025 * 0.99, 0.0025 * 0.99],
  [0.2, 0.2, 0.2, 0.2, 0.2],
  [0.2, 0.2, 0.2, 0.2, 0.2],
  [0.2, 0.2, 0.2, 0.2, 0.2],
  [0.2, 0.2, 0.2, 0.2, 0.2],
], ndmin = 2)

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

delta = 1
Q = 1 * np.identity(5)
```

```
x = np.array([
  [4],
  [4],
  [4],
  [4],
  [4]
], ndmin = 2)

K = np.zeros((5, 5))
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


do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 20)
```



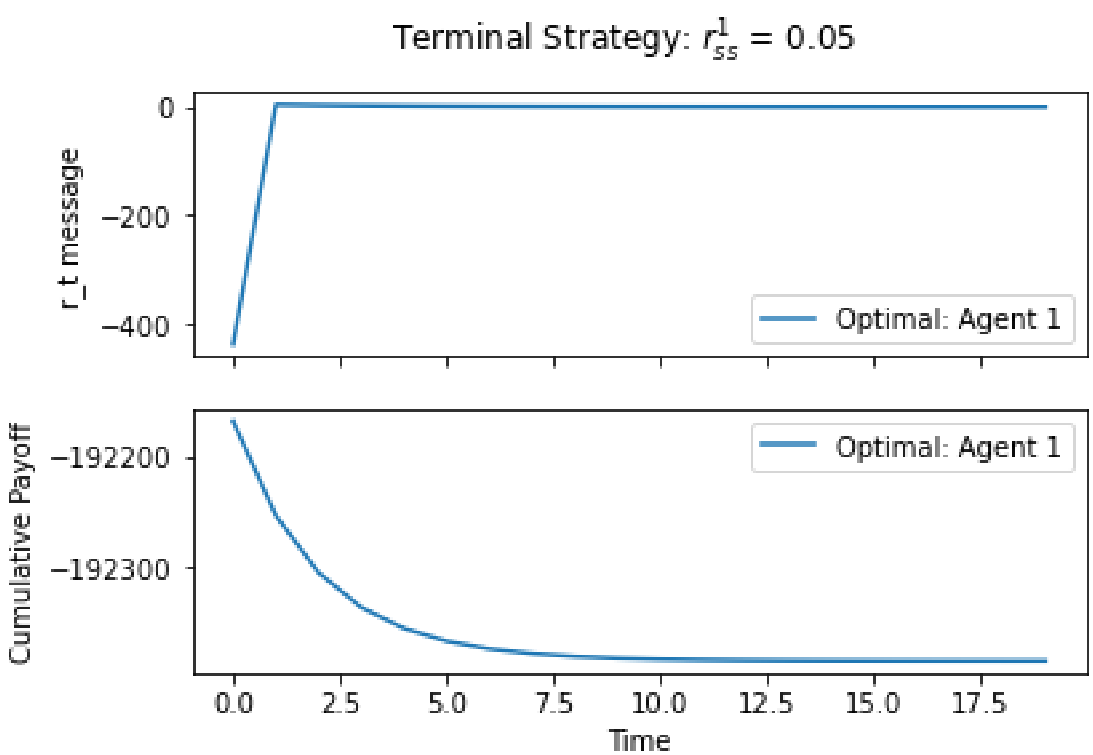
[46]:
```
K_t[0]
```

[46]:
```
array([[1.31231056, 0.31231056, 0.31231056, 0.31231056, 0.31231056],
       [0.31231056, 1.31231056, 0.31231056, 0.31231056, 0.31231056],
       [0.31231056, 0.31231056, 1.31231056, 0.31231056, 0.31231056],
       [0.31231056, 0.31231056, 0.31231056, 1.31231056, 0.31231056],
       [0.31231056, 0.31231056, 0.31231056, 0.31231056, 1.31231056]])
```

Note that this is very very similar to the case with only one easily-influenced agent. This case uses $R = 0$. The big difference is the messages are further away from zero.

[47]:
```
K = np.zeros((5, 5))
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

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*r_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


do_plot({0:r_ts}, [0], {0:payoffs}, num_agents = 1, set_cap = 20)
```



[48]:
```
r_ts[0].item()**2
```

[48]:
```
192088.66466466917
```

This is with $R = 1$. Payoff is extremely bad because of the extremely large $t = 0$ message being factored into quadratic cost.

[49]:
```
print(K_t[0])
```

```
[[132.55617878   3.50273146   3.50273146   3.50273146   3.50273146]
 [  3.50273146   1.48254381   0.48254381   0.48254381   0.48254381]
 [  3.50273146   0.48254381   1.48254381   0.48254381   0.48254381]
 [  3.50273146   0.48254381   0.48254381   1.48254381   0.48254381]
 [  3.50273146   0.48254381   0.48254381   0.48254381   1.48254381]]
```

There's some extra structural symmetry in the matrix.