Two_Cases_Targeting

October 4, 2021

1 Quick check of the basic targeting model

James Yu, 4 October 2021

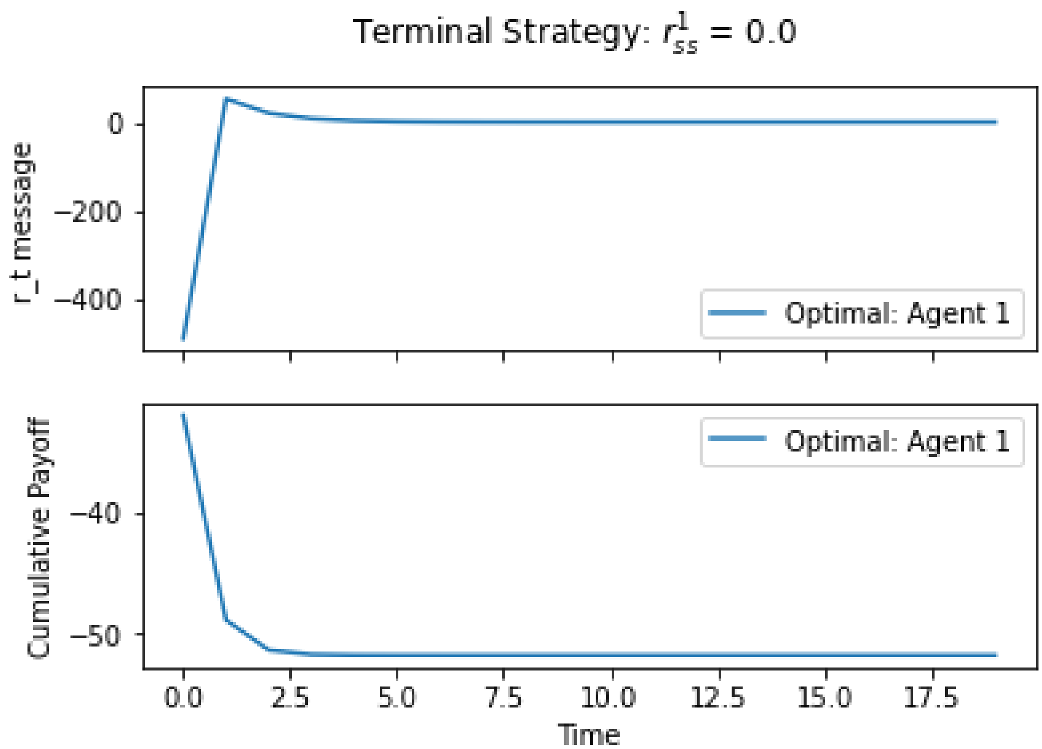
[1]:
```python
from collections import defaultdict
import matplotlib.pyplot as plt
import numpy as np
```

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

1.1 Case 1: target stubborn agent

[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


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



[4]:
```
print(K_t[0])
```

```
[[1.30901699 0.30901699]
 [0.30901699 1.30901699]]
```

Note this is the exact same K_{ss} from the symbolic notebook. Payoff is:

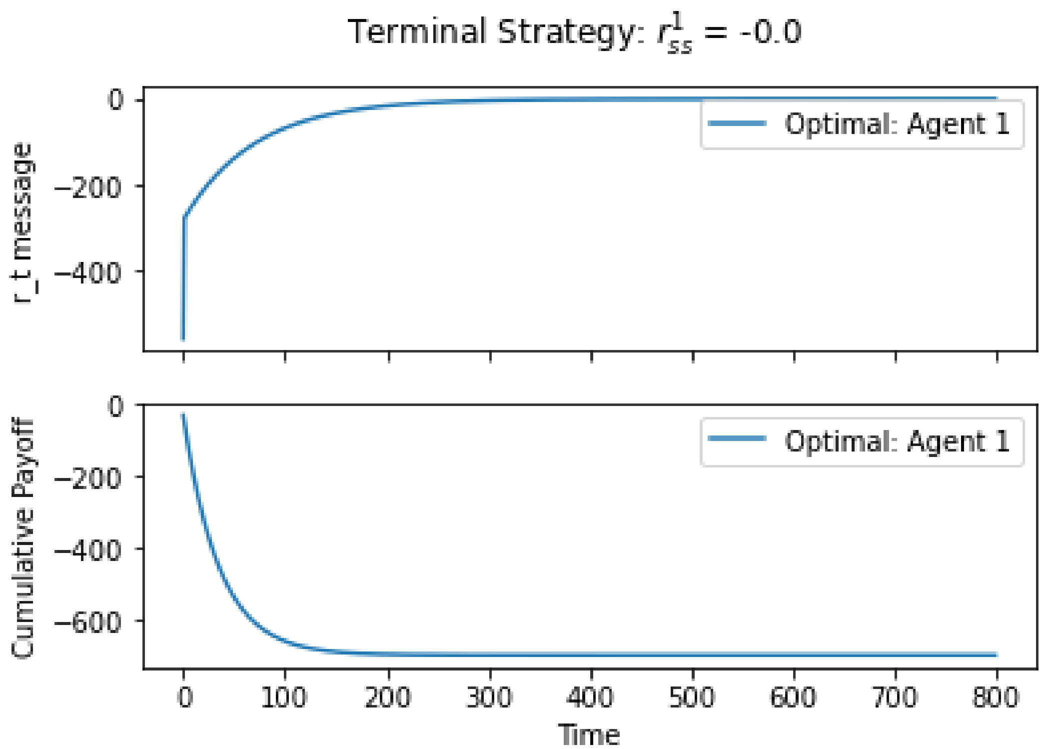
[5]:
```
payoffs[-1]
```

[5]: `-51.777087639996616`

1.2 Case 2: target easily-influenced agent

[6]:
```
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


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



It takes a far longer time for this model to converge. Payoff is clearly worse as well.

[7]:
```
payoffs[-1]
```

[7]: -699.4564175217757

[8]:
```
K_t[0]
```

[8]:
```
array([[41.88587718,  0.41298866],
       [ 0.41298866,  1.0041716 ]])
```

This also matches the symbolic notebook.