Repaired_Equations

September 24, 2021

1 Debugging the multi-agent equations

James Yu, 24 September 2021

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

1.1 Primary Code

Something in here is incorrect, and we need to figure out what it is by debugging the expressions at various points.

[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:
    template = [(B[l].T @ K[l] @ B[l]).item() for l in range(L)]
    base = [template.copy() 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)
            if 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
```

[6]:
```
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()
```

1.2 The model

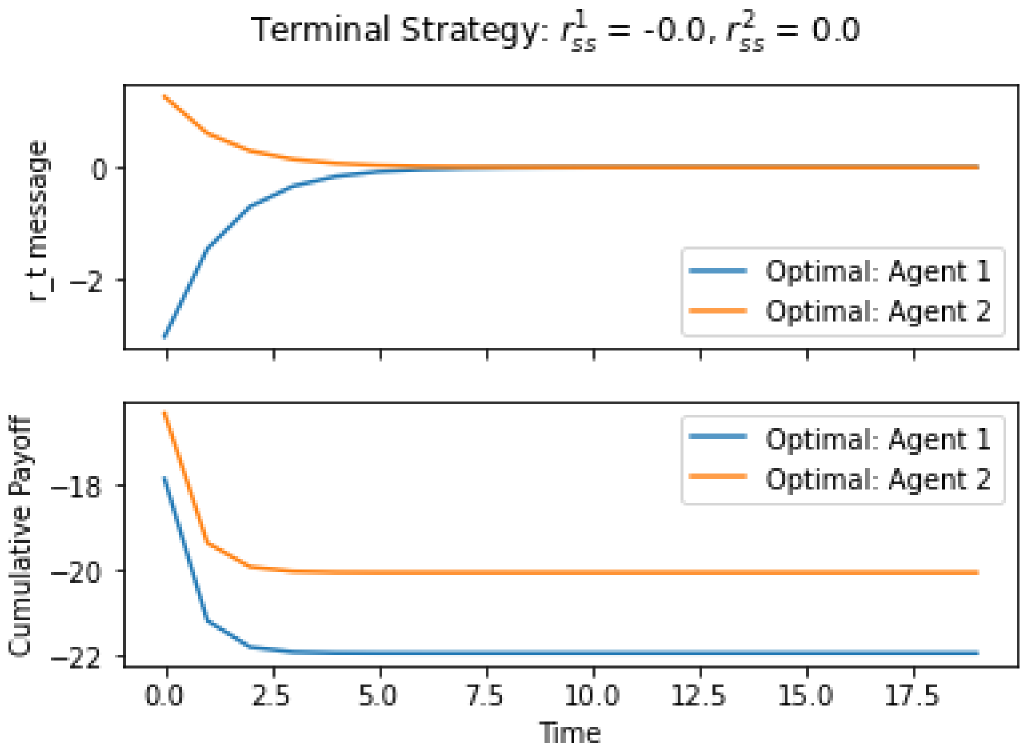
This is the model that displays incorrectly.

[7]:
```
A = np.array([
  [0.7],
], ndmin = 2)

B_1 = np.array([
  0.29, # split the channel of the strategic agent from the previous model␣
 ↪amongst the two strategic agents here
], ndmin = 2).T

B_2 = np.array([
```

```
  0.01, # split the channel of the strategic agent from the previous model␣
↪amongst the two strategic agents here
], ndmin = 2).T

B = [B_1, B_2]

X_0_1 = np.array([
  4,
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 1
m = 1
L = 2 # two agents now
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 0] # identical agents
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)]
```

[8]:
```
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)
```

[9]:
```
xs2, rs2, payoffs2 = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs2, r, payoffs2, num_agents = 2, set_cap = 20)
```



Observe the strategy of Agent 2 is positive, even though both agents have an agenda of zero. This is not supposed to happen.

In the analytical notebook, one of the first results was the sequence of optimal K_t. Here, that sequence is:

[10]:
```
arr = historical_K.copy()
arr.reverse()
print("\n".join(f"K_1 = {i[0].item()}, K_2 = {i[1].item()}" for i in arr))
```

```
K_1 = 1.0, K_2 = 1.0
K_1 = 1.2948056933938414, K_2 = 1.2271761924961753
K_1 = 1.3564445685532078, K_2 = 1.2523207726227819
K_1 = 1.3683629258980858, K_2 = 1.2536479315241391
K_1 = 1.3706330706765406, K_2 = 1.2533024555059933
K_1 = 1.3710643777654896, K_2 = 1.2531300771316545
K_1 = 1.3711463080157111, K_2 = 1.2530779978667934
K_1 = 1.371161875415893, K_2 = 1.253064588280239
K_1 = 1.3711648343304828, K_2 = 1.2530614004844431
K_1 = 1.371165396922692, K_2 = 1.2530606781905467
K_1 = 1.3711655039254147, K_2 = 1.2530605196955278
K_1 = 1.3711655242831644, K_2 = 1.253060485703953
K_1 = 1.371165528157458, K_2 = 1.253060478537978
K_1 = 1.371165528894984, K_2 = 1.2530604770472586
```

```
K_1 = 1.3711655290354199, K_2 = 1.253060476740424
K_1 = 1.3711655290621678, K_2 = 1.2530604766778128
K_1 = 1.3711655290672635, K_2 = 1.253060476665128
K_1 = 1.3711655290682345, K_2 = 1.2530604766625737
K_1 = 1.3711655290684197, K_2 = 1.2530604766620619
K_1 = 1.371165529068455, K_2 = 1.2530604766619597
K_1 = 1.371165529068462, K_2 = 1.2530604766619393
K_1 = 1.371165529068463, K_2 = 1.2530604766619353
K_1 = 1.3711655290684635, K_2 = 1.2530604766619349
K_1 = 1.3711655290684632, K_2 = 1.2530604766619347
K_1 = 1.3711655290684632, K_2 = 1.2530604766619344
K_1 = 1.3711655290684632, K_2 = 1.2530604766619344
```

In the semi-analytical exercise, the first K_1 and K_2 were some slightly lower numbers.

However, upon substituting the K_1 and K_2 from these equations into the analytical exercise, they turned out to not match either.

This means there is another problem beyond K_1 and K_2.

Optimal solution iteration comes from cell [5]. In particular, we need to check r_0^2, which is the one that is positive here and negative in the analytical.

[11]:
```
def optimal_find_problem(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)
    K_t = historical_K[0]
    k_t = historical_k[0]
    M_ = M(K_t, B, R, L, delta)
    print("M = ", M_)
    H_ = H(B, K_t, A, L)
    print("H = ", H_)
    E_ = E(M_, H_)
    print("E = ", E_)
    G_ = G(A, B, E_, L)
    print("G = ", G_)
    F_ = [F(M_, C_l(B, K_t, k_t, l, L, c, x, n), l) for l in range(L)]
    print("F = ", F_)
    g = [g_l(B, E_, h, x, F_, L) for h in range(L)]
    print("g = ", g)
    for l in range(L):
        print(-1 * E_[l:l+1, :] @ X_t[l] - F(M_, C_l(B, K_t, k_t, l, L, c, x,␣
 ↪n), l))
```

```
optimal_find_problem(X_0, historical_K, historical_k, historical_kappa)
```

```
M =  [[3.65315021e-01 1.25306048e-04]
 [1.15315021e-01 2.50125306e-01]]
H =  [[0.2783466 ]
 [0.00877142]]
E =  [[ 0.7620443 ]
 [-0.31625641]]
G =  [[0.48216972]]
F =  [array([[0.]]), array([[0.]])]
g =  [array([[0.]]), array([[0.]])]
[[-3.04817722]]
[[1.26502565]]
```

Let's compute each component step by step using the equations in the notes. We have $\gamma^2_{t-1} = -E^2_{t-1}\chi^2_{t-1}$.

First of all, M_{t-1} is going to be 2 by 2 here, where:

$$M_{t-1} = \begin{bmatrix} (b_1^2 K_t^1 + R_1\delta) & b_1 b_2 K_t^1 \\ b_1 b_2 K_t^2 & (b_2^2 K_t^2 + R_2\delta) \end{bmatrix}$$

We compute this as:

[12]:
```
K_1 = historical_K[0][0].item()
K_2 = historical_K[0][1].item()
b_1 = 0.29
b_2 = 0.01
R_me = 0.2
delta_me = 0.8
upper_left = b_1*b_1*K_1 + R_me/delta_me
upper_right = b_1*b_2*K_1
lower_left = b_1*b_2*K_2
lower_right = b_2*b_2*K_2 + R_me/delta_me
print([upper_left, upper_right])
print([lower_left, lower_right])
```

```
[0.36531502099465774, 0.003976380034298543]
[0.0036338753823196095, 0.2501253060476662]
```

This already doesn't match our M from before, so we take a look at this.

[13]:
```
def optimal_M_mistake(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)
    K_t = historical_K[0]
    k_t = historical_k[0]
    print(K_t, B, R, L, delta)
    M_ = M(K_t, B, R, L, delta)


optimal_M_mistake(X_0, historical_K, historical_k, historical_kappa)
```

```
[array([[1.37116553]]), array([[1.25306048]])] [array([[0.29]]),
array([[0.01]])] [array([[0.2]]), array([[0.2]])] 2 0.8
```

Everything appears ok going into the function for M so let's look at this specifically.

[14]:
```
def M_wrong(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:
    template = [(B[l].T @ K[l] @ B[l]).item() for l in range(L)]
    base = [template.copy() 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)
```

Actually, I can stop here because I found the problem. Observe that we use B_l in each row. However, this is clearly wrong, because we want products of every pair. The correct formulation is therefore:

[15]:
```
def M_corrected(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)
```

[16]:
```
def optimal_check_correct(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)
    K_t = historical_K[0]
    k_t = historical_k[0]
```

```
    M_ = M_corrected(K_t, B, R, L, delta)
    print("M = ", M_)
    H_ = H(B, K_t, A, L)
    print("H = ", H_)
    E_ = E(M_, H_)
    print("E = ", E_)
    G_ = G(A, B, E_, L)
    print("G = ", G_)
    F_ = [F(M_, C_l(B, K_t, k_t, l, L, c, x, n), l) for l in range(L)]
    print("F = ", F_)
    g = [g_l(B, E_, h, x, F_, L) for h in range(L)]
    print("g = ", g)
    for l in range(L):
        print(-1 * E_[l:l+1, :] @ X_t[l] - F(M_, C_l(B, K_t, k_t, l, L, c, x,␣
↪n), l))

optimal_check_correct(X_0, historical_K, historical_k, historical_kappa)
```

```
M =  [[0.36531502 0.00397638]
 [0.00363388 0.25012531]]
H =  [[0.2783466 ]
 [0.00877142]]
E =  [[0.76167457]
 [0.02400234]]
G =  [[0.47887435]]
F =  [array([[0.]]), array([[0.]])]
g =  [array([[0.]]), array([[0.]])]
[[-3.04669826]]
[[-0.09600936]]
```

This was the correct M:

[17]:
```
print([upper_left, upper_right])
print([lower_left, lower_right])
```

```
[0.36531502099465774, 0.003976380034298543]
[0.0036338753823196095, 0.2501253060476662]
```

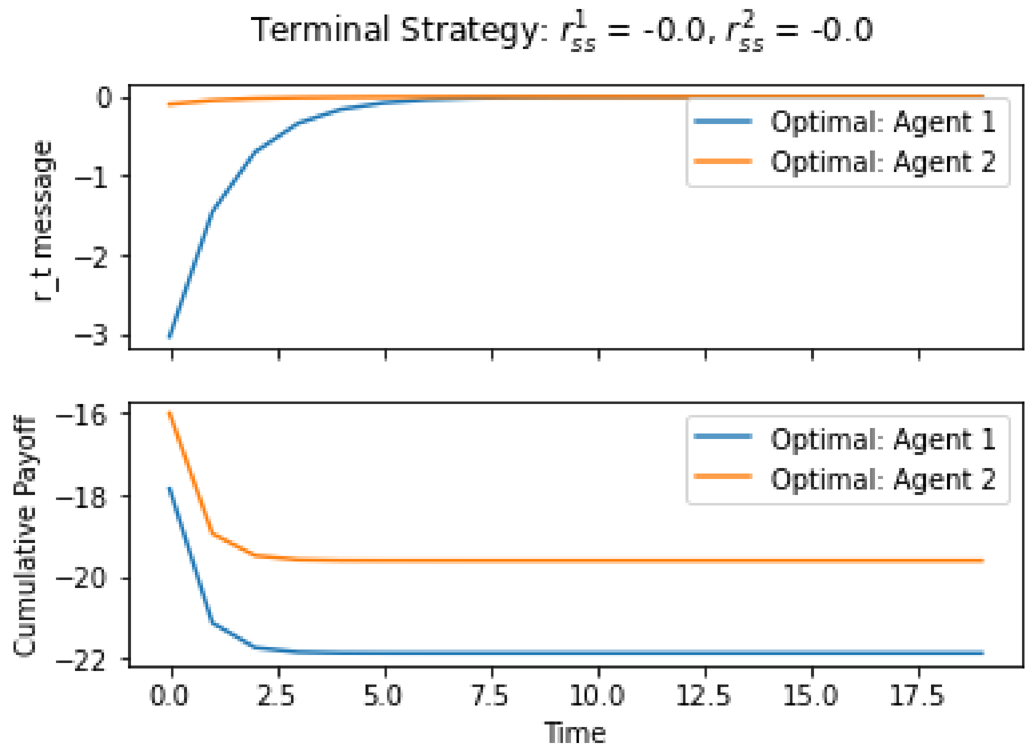
So this is now fixed, but r_0 is still slightly off.

It would be worthwhile to determine why K_t was slightly off initially. We check what we have fixed so far:

[22]:
```
def M(K, B, R, L, delta): # overwriting the wrong version with the right version
    """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)
```

[23]:
```
A = np.array([
  [0.7],
], ndmin = 2)

B_1 = np.array([
  0.29, # split the channel of the strategic agent from the previous model␣
 ↪amongst the two strategic agents here
], ndmin = 2).T

B_2 = np.array([
  0.01, # split the channel of the strategic agent from the previous model␣
 ↪amongst the two strategic agents here
], ndmin = 2).T

B = [B_1, B_2]

X_0_1 = np.array([
  4,
], ndmin = 2).T
X_0 = [X_0_1, X_0_1]

delta = 0.8
n = 1
m = 1
L = 2 # two agents now
Q = [1 * np.identity(n), 1 * np.identity(n)]
R = [0.2 * np.identity(m), 0.2 * np.identity(m)]

x = [0, 0] # identical agents
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)]

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)
xs2, rs2, payoffs2 = optimal(X_0, historical_K, historical_k, historical_kappa)
do_plot(rs2, r, payoffs2, num_agents = 2, set_cap = 20)
```



This looks ok, but let's check K:

[24]:
```
historical_K
```

[24]:
```
[[array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677856]]), array([[1.22537067]])],
 [array([[1.36677855]]), array([[1.22537068]])],
 [array([[1.36677854]]), array([[1.22537076]])],
 [array([[1.36677846]]), array([[1.2253711]])],
 [array([[1.36677802]]), array([[1.22537275]])],
 [array([[1.36677562]]), array([[1.22538032]])],
 [array([[1.36676266]]), array([[1.22541419]])],
```

```
 [array([[1.3666923]]), array([[1.22555862]])],
 [array([[1.36631033]]), array([[1.2261282]])],
 [array([[1.36423671]]), array([[1.22805052]])],
 [array([[1.35300682]]), array([[1.23208755]])],
 [array([[1.29314984]]), array([[1.2194456]])],
 [array([[1.]]), array([[1.]])]]
```

This actually matches much closer than before. r_0 is:

[26]:
```
rs2[0][0]
```

[26]:
```
array([[-3.04004274]])
```

[28]:
```
rs2[1][0]
```

[28]:
```
array([[-0.09398337]])
```

This is still wrong however because we expected r_0^2 to be about -0.18. So we keep searching.