StrategicInfluenceProof

July 13, 2021

1 Numerical Verification of Formulas from 3.4 Derivation

James Yu, 12 July 2021

[1]:
```
import numpy as np
```

In this notebook I test-drive a single period of iteration of the hypothesized optimal coefficient matrices. To do this, I determine K_{T-1}, k_{T-1}, κ_{T-1} and γ^l_{T-1} given all known values, and then plug everything back into the Bellman to ensure they are correct.

First, I set up all variables. For simplicity, I use $n = 2$, $L = 2$, and $m_l = 1, \forall l$.

[2]:
```
A = np.array([
    [0.25, 0.25],
    [0.25, 0.25]
], ndmin = 2)

B_1 = np.array([
    [0.25],
    [0.25]
], ndmin = 2)

B_2 = np.array([
    [0.25],
    [0.25]
], ndmin = 2)

delta = 0.8

Q_1 = 0.2 * np.identity(2) # the same Q for both strategic agents, for now
Q_2 = Q_1
R_1 = 0.2 * np.identity(1) # the same R as well
R_2 = R_1

K_t_1 = Q_1
K_t_2 = Q_2

M_t_minus_1 = np.array([
    [(B_1.T @ K_t_1 @ B_1 + R_1/delta).item(), (B_1.T @ K_t_1 @ B_2).item()],
```

```
    [(B_2.T @ K_t_2 @ B_1).item(), (B_2.T @ K_t_2 @ B_2 + R_2/delta).item()]
], ndmin = 2)

H_t_minus_1 = np.concatenate((
    B_1.T @ K_t_1 @ A,
    B_2.T @ K_t_2 @ A
), axis = 0)

r_1 = 0
r_2 = 10
x_1 = 0
x_2 = 10
c_base = B_1 @ np.array([[r_1]], ndmin = 2) + B_2 @ np.array([[r_2]], ndmin = 2)
c_1 = c_base + (A - np.identity(2)) @ (x_1 * np.ones((2, 1)))
c_2 = c_base + (A - np.identity(2)) @ (x_2 * np.ones((2, 1)))


C_t_minus_1_1 = np.concatenate(( # this only solves for agent 1; separate one␣
 ↪needed for agent 2
    B_1.T @ K_t_1 @ (A @ ((x_1 - x_1) * np.ones((2, 1))) + c_1) + 0, # 0.5 *␣
 ↪B_1.T @ k
    B_2.T @ K_t_2 @ (A @ ((x_1 - x_2) * np.ones((2, 1))) + c_1) + 0
), axis = 0)

C_t_minus_1_2 = np.concatenate(( # this only solves for agent 2
    B_1.T @ K_t_1 @ (A @ ((x_2 - x_1) * np.ones((2, 1))) + c_1) + 0, # 0.5 *␣
 ↪B_1.T @ k
    B_2.T @ K_t_2 @ (A @ ((x_2 - x_2) * np.ones((2, 1))) + c_1) + 0
), axis = 0)
```

[3]:
```
E_t_minus_1 = np.linalg.inv(M_t_minus_1) @ H_t_minus_1
E_t_minus_1_1 = np.array(E_t_minus_1[0, :], ndmin = 2)
E_t_minus_1_2 = np.array(E_t_minus_1[1, :], ndmin = 2)
E_t_minus_1
```

[3]:
```
array([[0.08333333, 0.08333333],
       [0.08333333, 0.08333333]])
```

[4]:
```
F_t_minus_1_1 = np.array((np.linalg.inv(M_t_minus_1) @ C_t_minus_1_1)[0, :],␣
 ↪ndmin = 2)
F_t_minus_1_2 = np.array((np.linalg.inv(M_t_minus_1) @ C_t_minus_1_2)[1, :],␣
 ↪ndmin = 2)
F_t_minus_1_1
```

[4]:
```
array([[1.]])
```

[5]:
```
G_t_minus_1 = A - (B_1 @ E_t_minus_1_1 + B_2 @ E_t_minus_1_2)
```

[6]:
```
g_t_minus_1_1 = c_1 - (B_1 @ (E_t_minus_1_1 @ ((x_1 - x_1) * np.ones((2, 1))) +␣
 ↪F_t_minus_1_1) + B_2 @ (E_t_minus_1_2 @ ((x_1 - x_2) * np.ones((2, 1))) +␣
 ↪F_t_minus_1_2))
```

First, K^1_{t-1}:

[7]:
```
K_t_minus_1_1 = Q_1 + E_t_minus_1_1.T @ R_1 @ E_t_minus_1_1 + delta *␣
 ↪G_t_minus_1.T @ K_t_1 @ G_t_minus_1
K_t_minus_1_1
```

[7]:
```
array([[0.21527778, 0.01527778],
       [0.01527778, 0.21527778]])
```

Next, k^1_{t-1}:

[8]:
```
k_t_minus_1_1 = 2*delta* g_t_minus_1_1.T @ K_t_1 @ G_t_minus_1 + 2 *␣
 ↪F_t_minus_1_1.T @ R_1 @ E_t_minus_1_1
k_t_minus_1_1
```

[8]:
```
array([[0.36666667, 0.36666667]])
```

Finally, κ^l_{t-1}:

[9]:
```
kappa_t_minus_1_1 = -delta * (g_t_minus_1_1.T @ K_t_1 @ g_t_minus_1_1) -␣
 ↪(F_t_minus_1_1.T @ R_1 @ F_t_minus_1_1)
kappa_t_minus_1_1
```

[9]:
```
array([[-2.2]])
```

Now, verify using the Bellman and some random χ^l_{t-1}.

[10]:
```
chi_t_minus_1_1 = np.array([
    0,
    4
], ndmin = 2).T
```

[11]:
```
gamma_t_minus_1_1 = -E_t_minus_1_1 @ chi_t_minus_1_1 - F_t_minus_1_1
gamma_t_minus_1_1
```

[11]:
```
array([[-1.33333333]])
```

[12]:
```
chi_t_1 = G_t_minus_1 @ chi_t_minus_1_1 + g_t_minus_1_1
```

[13]:
```
RHS = -chi_t_minus_1_1.T @ Q_1 @ chi_t_minus_1_1 - gamma_t_minus_1_1.T @ R_1 @␣
 ↪gamma_t_minus_1_1 + delta * (-chi_t_1.T @ Q_1 @ chi_t_1)
RHS
```

[13]:
```
array([[-7.11111111]])
```

[14]:
```
LHS = -chi_t_minus_1_1.T @ K_t_minus_1_1 @ chi_t_minus_1_1 - k_t_minus_1_1 @␣
 ↪chi_t_minus_1_1 + kappa_t_minus_1_1
LHS
```

[14]:
```
array([[-7.11111111]])
```

Clearly RHS = LHS. To test the other matrices requires an additional period as follows. First, solve for the other strategic agent:

[15]:
```
g_t_minus_1_2 = c_2 - (B_1 @ (E_t_minus_1_1 @ ((x_2 - x_1) * np.ones((2, 1))) +␣
 ↪F_t_minus_1_1) + B_2 @ (E_t_minus_1_2 @ ((x_2 - x_2) * np.ones((2, 1))) +␣
 ↪F_t_minus_1_2))
```

[16]:
```
K_t_minus_1_2 = Q_2 + E_t_minus_1_2.T @ R_2 @ E_t_minus_1_2 + delta *␣
 ↪G_t_minus_1.T @ K_t_2 @ G_t_minus_1
K_t_minus_1_2
```

[16]:
```
array([[0.21527778, 0.01527778],
       [0.01527778, 0.21527778]])
```

[17]:
```
k_t_minus_1_2 = 2*delta* g_t_minus_1_2.T @ K_t_2 @ G_t_minus_1 + 2 *␣
 ↪F_t_minus_1_2.T @ R_2 @ E_t_minus_1_2
k_t_minus_1_2
```

[17]:
```
array([[-0.42222222, -0.42222222]])
```

[18]:
```
kappa_t_minus_1_2 = -delta * (g_t_minus_1_2.T @ K_t_2 @ g_t_minus_1_2) -␣
 ↪(F_t_minus_1_2.T @ R_2 @ F_t_minus_1_2)
kappa_t_minus_1_2
```

[18]:
```
array([[-3.64444444]])
```

Now check, just in case:

[19]:
```
chi_t_minus_1_2 = chi_t_minus_1_1
gamma_t_minus_1_2 = -E_t_minus_1_2 @ chi_t_minus_1_2 - F_t_minus_1_2
gamma_t_minus_1_2
```

[19]:
```
array([[-1.]])
```

[20]:
```
chi_t_2 = G_t_minus_1 @ chi_t_minus_1_2 + g_t_minus_1_2
```

[21]:
```
RHS = -chi_t_minus_1_2.T @ Q_2 @ chi_t_minus_1_2 - gamma_t_minus_1_2.T @ R_2 @␣
 ↪gamma_t_minus_1_2 + delta * (-chi_t_2.T @ Q_2 @ chi_t_2)
RHS
```

[21]:
```
array([[-5.4]])
```

[22]:
```
LHS = -chi_t_minus_1_2.T @ K_t_minus_1_2 @ chi_t_minus_1_2 - k_t_minus_1_2 @␣
 ↪chi_t_minus_1_2 + kappa_t_minus_1_2
LHS
```

[22]:
```
array([[-5.4]])
```

Also exactly the same. Now proceed:

[23]:
```
M_t_minus_2 = np.array([
    [(B_1.T @ K_t_minus_1_1 @ B_1 + R_1/delta).item(), (B_1.T @ K_t_minus_1_1 @␣
 ↪B_2).item()],
    [(B_2.T @ K_t_minus_1_2 @ B_1).item(), (B_2.T @ K_t_minus_1_2 @ B_2 + R_2/
 ↪delta).item()]
], ndmin = 2)

H_t_minus_2 = np.concatenate((
```

```
    B_1.T @ K_t_minus_1_1 @ A,
    B_2.T @ K_t_minus_1_2 @ A
), axis = 0)

C_t_minus_2_1 = np.concatenate(( # this only solves for agent 1; separate one␣
 ↪needed for agent 2
    B_1.T @ K_t_minus_1_1 @ (A @ ((x_1 - x_1) * np.ones((2, 1))) + 0.5 * B_1.T␣
 ↪@ k_t_minus_1_1.T + c_1) + 0, # 0.5 * B_1.T @ k
    B_2.T @ K_t_minus_1_2 @ (A @ ((x_1 - x_2) * np.ones((2, 1))) + 0.5 * B_1.T␣
 ↪@ k_t_minus_1_2.T + c_1) + 0
), axis = 0)

C_t_minus_2_2 = np.concatenate(( # this only solves for agent 2
    B_1.T @ K_t_minus_1_1 @ (A @ ((x_2 - x_1) * np.ones((2, 1))) + 0.5 * B_1.T␣
 ↪@ k_t_minus_1_1.T + c_1) + 0, # 0.5 * B_1.T @ k
    B_2.T @ K_t_minus_1_2 @ (A @ ((x_2 - x_2) * np.ones((2, 1))) + 0.5 * B_1.T␣
 ↪@ k_t_minus_1_2.T + c_1) + 0
), axis = 0)
```

[24]:
```
E_t_minus_2 = np.linalg.inv(M_t_minus_2) @ H_t_minus_2
E_t_minus_2_1 = np.array(E_t_minus_2[0, :], ndmin = 2)
E_t_minus_2_2 = np.array(E_t_minus_2[1, :], ndmin = 2)
E_t_minus_2
```

[24]:
```
array([[0.09367946, 0.09367946],
       [0.09367946, 0.09367946]])
```

[25]:
```
F_t_minus_2_1 = np.array((np.linalg.inv(M_t_minus_2) @ C_t_minus_2_1)[0, :],␣
 ↪ndmin = 2)
F_t_minus_2_2 = np.array((np.linalg.inv(M_t_minus_2) @ C_t_minus_2_2)[1, :],␣
 ↪ndmin = 2)
F_t_minus_2_1
```

[25]:
```
array([[1.19564625]])
```

[26]:
```
G_t_minus_2 = A - (B_1 @ E_t_minus_2_1 + B_2 @ E_t_minus_2_2)
```

[27]:
```
g_t_minus_2_1 = c_1 - (B_1 @ (E_t_minus_2_1 @ ((x_1 - x_1) * np.ones((2, 1))) +␣
 ↪F_t_minus_2_1) + B_2 @ (E_t_minus_2_2 @ ((x_1 - x_2) * np.ones((2, 1))) +␣
 ↪F_t_minus_2_2))
```

[28]:
```
K_t_minus_2_1 = Q_1 + E_t_minus_2_1.T @ R_1 @ E_t_minus_2_1 + delta *␣
 ↪G_t_minus_2.T @ K_t_minus_1_1 @ G_t_minus_2
K_t_minus_2_1
```

[28]:
```
array([[0.21698072, 0.01698072],
       [0.01698072, 0.21698072]])
```

[29]:
```
k_t_minus_2_1 = 2*delta* g_t_minus_2_1.T @ K_t_minus_1_1 @ G_t_minus_2 + delta␣
 ↪* k_t_minus_1_1 @ G_t_minus_2 + 2 * F_t_minus_2_1.T @ R_1 @ E_t_minus_2_1
k_t_minus_2_1
```

[29]:
```
array([[0.53890321, 0.53890321]])
```

Note the new term:

[30]:
```
delta * k_t_minus_1_1 @ G_t_minus_2
```

[30]:
```
array([[0.11918736, 0.11918736]])
```

Continuing,

[31]:
```
kappa_t_minus_2_1 = -delta * (g_t_minus_2_1.T @ K_t_minus_1_1 @ g_t_minus_2_1 +␣
 ↪k_t_minus_1_1 @ g_t_minus_2_1 - kappa_t_minus_1_1) - (F_t_minus_2_1.T @ R_1␣
 ↪@ F_t_minus_2_1)
kappa_t_minus_2_1
```

[31]:
```
array([[-5.82129996]])
```

Now, verify with a random χ (doesn't need to be related to the original, can even be the same as the original because the Bellman should work with anything):

[32]:
```
chi_t_minus_2_1 = chi_t_minus_1_1
gamma_t_minus_2_1 = -E_t_minus_2_1 @ chi_t_minus_2_1 - F_t_minus_2_1
gamma_t_minus_2_1
```

[32]:
```
array([[-1.57036408]])
```

[33]:
```
chi_t_minus_1_1_again = G_t_minus_2 @ chi_t_minus_2_1 + g_t_minus_2_1
```

[34]:
```
RHS = -chi_t_minus_2_1.T @ Q_1 @ chi_t_minus_2_1 - gamma_t_minus_2_1.T @ R_1 @␣
 ↪gamma_t_minus_2_1 + delta * (-chi_t_minus_1_1_again.T @ K_t_minus_1_1 @␣
 ↪chi_t_minus_1_1_again - k_t_minus_1_1 @ chi_t_minus_1_1_again +␣
 ↪kappa_t_minus_1_1)
RHS
```

[34]:
```
array([[-11.44860437]])
```

[35]:
```
LHS = -chi_t_minus_2_1.T @ K_t_minus_2_1 @ chi_t_minus_2_1 - k_t_minus_2_1 @␣
 ↪chi_t_minus_2_1 + kappa_t_minus_2_1
LHS
```

[35]:
```
array([[-11.44860437]])
```

Exactly the same, as needed, meaning all three equations are correct. Note that if any part were to be wrong, this would break. Take for example:

[36]:
```
kappa_t_minus_2_1 = -delta * (g_t_minus_2_1.T @ K_t_minus_1_1 @ g_t_minus_2_1 +␣
 ↪k_t_minus_1_1 @ g_t_minus_2_1 + kappa_t_minus_1_1) - (F_t_minus_2_1.T @ R_1␣
 ↪@ F_t_minus_2_1)
kappa_t_minus_2_1
```

[36]:
```
array([[-2.30129996]])
```

In the above expression I replaced the - in front of kappa_t_minus_1_1 with a + (a typo from the notes at one point). This is what happens (note no change to χ, which is independent):

[37]:
```

```

```
RHS = -chi_t_minus_2_1.T @ Q_1 @ chi_t_minus_2_1 - gamma_t_minus_2_1.T @ R_1 @␣
 ↪gamma_t_minus_2_1 + delta * (-chi_t_minus_1_1_again.T @ K_t_minus_1_1 @␣
 ↪chi_t_minus_1_1_again - k_t_minus_1_1 @ chi_t_minus_1_1_again +␣
 ↪kappa_t_minus_1_1)
RHS
```

[37]:
```
array([[-11.44860437]])
```

[38]:
```
LHS = -chi_t_minus_2_1.T @ K_t_minus_2_1 @ chi_t_minus_2_1 - k_t_minus_2_1 @␣
 ↪chi_t_minus_2_1 + kappa_t_minus_2_1
LHS
```

[38]:
```
array([[-7.92860437]])
```

The RHS is still the same because it is dependent on prior-period information. But the LHS, based on the new information, is wrong because of the typo that was made. Thus, the original equations must not have any typos in them that mathematically impact the recursive equations.